UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/690,866	10/22/2003	Michael J. Wookey	30014200-1100	6923

58328 7590 01/23/2008

SUN MICROSYSTEMS
C/O SONNENSCHEIN NATH & ROSENTHAL LLP
P.O. BOX 061080
WACKER DRIVE STATION, SEARS TOWER
CHICAGO, IL 60606-1080

EXAMINER	
STARKS, WILBERT L	

ART UNIT	PAPER NUMBER
2129	

MAIL DATE	DELIVERY MODE
01/23/2008	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/690,866	WOOKEY ET AL.	
	Examiner	Art Unit	
	Wilbert L. Starks, Jr.	2129	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *22 October 2003*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-74* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-74* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

Claim Rejections - 35 U.S.C. §101

1. 35 U.S.C. §101 reads as follows:

> Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title.

the invention as disclosed in claims 1-74 is directed to non-statutory subject matter.

2. None of the claims is limited to practical applications that indicate a specific practical utility for the claimed invention. Examiner finds that *In re Warmerdam*, 33 F.3d 1354, 31 USPQ2d 1754 (Fed. Cir. 1994) controls the 35 U.S.C. §101 issues on that point for reasons made clear by the Federal Circuit in *AT&T Corp. v. Excel Communications, Inc.*, 50 USPQ2d 1447 (Fed. Cir. 1999). Specifically, the Federal Circuit held that the act of:

> ...[T]aking several abstract ideas and manipulating them together adds nothing to the basic equation. *AT&T v. Excel* at 1453 quoting *In re Warmerdam*, 33 F.3d 1354, 1360 (Fed. Cir. 1994).

Examiner finds that Applicant's "input data" references are just such abstract ideas.

3. Examiner bases his position upon guidance provided by the Federal Circuit in *In re Warmerdam*, as interpreted by *AT&T v. Excel*. This set of precedents is within the same line of cases as the *Alappat-State Street Bank* decisions and is in complete

agreement with those decisions. *Warmerdam* is consistent with *State Street*'s holding that:

> Today we hold that *the transformation of data, representing discrete dollar amounts, by a machine through a series of mathematical calculations into a final share price*, constitutes a practical application of a mathematical algorithm, formula, or calculation because it produces 'a useful, concrete and tangible result" -- *a final share price momentarily fixed for recording purposes and even accepted and relied upon by regulatory authorities and in subsequent trades.* (emphasis added) *State Street Bank* at 1601.

4. True enough, that case later eliminated the "business method exception" in order to show that business methods were not per se nonstatutory, but the court clearly *did not* go so far as to make business methods *per se statutory*. A plain reading of the excerpt above shows that the Court was *very specific* in its definition of the new *practical application* that indicates a specific practical utility for the claimed invention. It would have been much easier for the court to say that "business methods were per se statutory" than it was to define the practical application in the case as "...the transformation of data, representing discrete dollar amounts, by a machine through a series of mathematical calculations into a final share price..."

5. The court was being very specific.

6. Additionally, the court was also careful to specify that the "useful, concrete and tangible result" it found was "a final share price momentarily fixed for recording purposes and even accepted and relied upon by regulatory authorities and in

subsequent trades." (i.e. the trading activity is the further practical use of the real world monetary data beyond the transformation in the computer – i.e., "post-processing activity".)

7. Applicant cites no such specific results to define a useful, concrete and tangible result. Neither does Applicant specify the associated practical application with the kind of specificity the Federal Circuit used.

8. Furthermore, in the case *In re Warmerdam*, the Federal Circuit held that:

> ...[T]he dispositive issue for assessing compliance with Section 101 in this case is whether the claim is for a process that goes beyond simply manipulating 'abstract ideas' or 'natural phenomena' ... As the Supreme Court has made clear, '[a]n idea of itself is not patentable, ... *taking several abstract ideas and manipulating them together adds nothing to the basic equation*. In re Warmerdam 31 USPQ2d at 1759 (emphasis added).

9. Since the Federal Circuit held in *Warmerdam* that this is the "dispositive issue" when it judged the usefulness, concreteness, and tangibility of the claim limitations in that case, Examiner in the present case views this holding as the dispositive issue for determining whether a claim is "useful, concrete, and tangible" in similar cases. Accordingly, the Examiner finds that Applicant manipulated a set of abstract "input data" to solve purely algorithmic problems in the abstract (i.e., what *kinds* of "input data" are used? Heart rhythm data? Algebraic equations? Boolean logic problems? Fuzzy logic algorithms? Probabilistic word problems? Philosophical ideas? Even vague expressions, about which even reasonable persons could differ as to their meaning? Combinations thereof?) Clearly, a claim for manipulation of "input data" is provably even more abstract (and thereby less limited in practical application) than pure "mathematical algorithms" which the Supreme Court has held are per se nonstatutory – in fact, it *includes* the expression of nonstatutory mathematical algorithms.

10. Since *Warmerdam* is within the *Alappat-State Street Bank* line of cases, it takes the same view of "useful, concrete, and tangible" the Federal Circuit applied in *State Street Bank*. Therefore, under *State Street Bank*, this could not be a "useful, concrete and tangible result". There is only manipulation of abstract ideas.

11. The Federal Circuit validated the use of *Warmerdam* in its more recent *AT&T Corp. v. Excel Communications, Inc.* decision. The Court reminded us that:

> Finally, **the decision in In re Warmerdam**, 33 F.3d 1354, 31 USPQ2d 1754 (Fed. Cir. 1994) **is not to the contrary**. *** The court found that the claimed process did nothing more than manipulate basic

> mathematical constructs and concluded that '**taking several abstract ideas and manipulating them together adds nothing to the basic equation**'; hence, the court held that the claims were properly rejected under §101 ... Whether one agrees with the court's conclusion on the facts, the **holding** of the case is a straightforward application of the basic principle that mere laws of nature, natural phenomena, and abstract ideas are not within the categories of inventions or discoveries that may be patented under §101. (emphasis added) AT&T Corp. v. Excel Communications, Inc., 50 USPQ2d 1447, 1453 (Fed. Cir. 1999).

12. Remember that in *In re Warmerdam*, the Court said that this was the dispositive issue to be considered. In the *AT&T* decision cited above, the Court reaffirms that this is the issue for assessing the "useful, concrete, and tangible" nature of a set of claims under §101 doctrine. Accordingly, Examiner views the *Warmerdam* holding as the dispositive issue in this analogous case.

13. The fact that the invention is merely the manipulation of *abstract ideas* is clear. The data referred to by Applicant's idea of "input data" is simply an abstract construct that does not provide limitations in the claims to the transformation of real world data (such as monetary data or heart rhythm data) by some disclosed process. Consequently, the necessary conclusion under *AT&T, State Street* and *Warmerdam,* is straightforward and clear. The claims take several abstract ideas (i.e., "input data" in the abstract) and manipulate them together adding nothing to the basic equation. Claims 1-74 are, thereby, rejected under 35 U.S.C. §101.

Claim Rejections - 35 U.S.C. §112

The following is a quotation of the first paragraph of 35 U.S.C. §112:

> The specification shall contain a written description of the invention, and of the manner and process of making and using it, in such full, clear, concise, and exact terms as to enable any person skilled in the

art to which it pertains, or with which it is most nearly connected, to make and use the same and shall set forth the best mode contemplated by the inventor of carrying out his invention.

Claims 1-74 are rejected under 35 U.S.C. §112, first paragraph because current case law (and accordingly, the MPEP) require such a rejection if a §101 rejection is given because when Applicant has not in fact disclosed the practical application for the invention, as a matter of law there is no way Applicant could have disclosed *how* to practice the *undisclosed* practical application. This is how the MPEP puts it:

> ("The how to use prong of section 112 **incorporates as a matter of law** the requirement of 35 U.S.C. §101 that the specification disclose as a matter of fact a practical utility for the invention.... If the application fails as a matter of fact to satisfy 35 U.S.C. §101, then the application **also fails as a matter of law** to enable one of ordinary skill in the art to use the invention under 35 U.S.C. §112."); In re Kirk, 376 F.2d 936, 942, 153 USPQ 48, 53 (CCPA 1967) ("Necessarily, compliance with §112 requires a description of how to use presently useful inventions, **otherwise an applicant would anomalously be required to teach how to use a useless invention**.") See, MPEP 2107.01(IV), quoting In re Kirk (emphasis added).

Examiner made a §101 utility rejection of the claims because they fail to indicate a specific practical utility (i.e., practical application) for the claimed invention. Therefore, claims 1-74 are rejected on this basis.

Claim Rejections - 35 U.S.C. §102

The following is a quotation of the appropriate paragraphs of 35 U.S.C. §102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –

(e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

Claims 1-74 are rejected under 35 U.S.C. §102(e) as being anticipated by Rice, et al. (U.S. Patent Number 6,973,560; dated 06 DEC 2005; class 712; subclass 205). Specifically:

Claim 1

Claim 1's "receiving a rule as input from a user," is anticipated by Rice, et al., column 10, lines 38-55, where it recites:

> Referring now to FIG. 3, master codon table 304 is shown containing ten (10) codons (numbered 1 through 10), as indicated by a codon number column 306 in the master codon table 304. The entity that each codon number specifically represents is in the "label" column 308. Thus, codon number 3 represents the "*" operation (i.e., multiplication), codon number 5 represents the constant "9", codon number 10 represents a function called "xlCell.RValue" (which is an abbreviation for Excel read value), etc. (In this example, the Excel program from Microsoft Corporation is mentioned, but any other functionally similar program could also have been used.) The codons shown and described in this Figure were specifically chosen to illustrate the implementation of the above-mentioned temperature conversion program, though it should be clearly understood that the present invention contemplates that the master codon table can contain any number and different types of **codons** (pre-set or **user-defined**).

Claim 1's "the rule defining a logic for determining exposure to failure of a computer-based system based on input data about the computer-based system;" is anticipated by Rice, et al., column 55, lines 15-30, where it recites:

> To perform self-modification and error handling requires state information ("introspection"). As a design philosophy, one may attempt to make all state information generated by the Virtual Machine in evaluating agents available to the agents themselves. Metaphorically, to correct errors or make adjustments, one needs to know "Where am I? What went wrong? What was the configuration at the time of the error?" Embodiments of the present invention envision that the virtual machine environment consists of an array of state records and some global variables (userCount, steps, errors, lasterror, etc.). All this information is available in read-only mode to the agents. Agents can also query information about the processes that spawned them, enabling context sensitive logic (if this process called me then do x, otherwise do y).

Claim 1's "preparing a rule datatype including the rule; and" is anticipated by Rice, et al., Abstract, where it recites:

> A fault tolerant software environment, in which various program components (e.g., portions of computer programs, applications, etc) are objectized into entities represented by "codons." This allows for improper syntax to occur, enabling, for example, combinatorial operations such as genetic programming. The present invention also contemplates such features as the ability to probabilistically execute individual codons, to switch between **treating information as executable code or as data** (or passing over it), provides that the individual codons can be tagged so that additional information can be associated with them, and provides for tagging of the stack.

Claim 1's "publishing the rule datatype to a network connected to the data processing system." is anticipated by Rice, et al., column 42, lines 36-52, where it recites:

> Client/Server **codons** allow for many possible **network** architectures, and embodiments are shown with regard to FIG. 20. Referring to FIG. 20, in the example depicted by that Figure, a request to perform a given task is initiated by a client computer X (2002) and sent to Computer A for processing. Using the features envisioned by the present invention, the request could then be routed from computer A to, depending upon load,

> resources, etc., one or more of computers B, C, D, etc. Assuming (in this example) that Computer D (2010) processes (or helps process) the request, **it is shown that Computer D then passes the result back to client computer X (2002).** In addition to this specific example, FIG. 20 also indicates that more than one client can be involved in a given transaction. It should be understood that the virtual machine environment is contemplated to contain or be contained in, one or more of the various computers/clients shown.

Claim 2

Claim 2's "issuing a query to the network requesting a subscriber identifier of a subscriber to the rule datatype; and" is anticipated by Rice, et al., column 55, lines 16-30, where it recites:

> To perform self-modification and error handling requires state information ("introspection"). As a design philosophy, one may attempt to make all state information generated by the Virtual Machine in evaluating agents available to the agents themselves. Metaphorically, to correct errors or make adjustments, one needs to know "Where am I? What went wrong? What was the configuration at the time of the error?" Embodiments of the present invention envision that the virtual machine environment consists of an array of state records and some global variables (userCount, steps, errors, lasterror, etc.). All this information is available in read-only mode to the agents. Agents can also **query information about the processes that spawned them**, enabling context sensitive logic (if this process called me then do x, otherwise do y).

Claim 2's "receiving the subscriber identifier responsive to the issued query, wherein the rule datatype includes the subscriber identifier." is anticipated by Rice, et al., column 55, lines 16-30, where it recites:

> To perform self-modification and error handling requires state information ("introspection"). As a design philosophy, one may attempt to make all state information generated by the Virtual Machine in evaluating agents available to the agents themselves. Metaphorically, to correct errors or make adjustments, one needs to know "Where am I? What went wrong? What was the configuration at the time of the error?" Embodiments of the

> present invention envision that the virtual machine environment consists of an array of state records and some global variables (userCount, steps, errors, lasterror, etc.). All this information is available in read-only mode to the agents. Agents can also **query information about the processes that spawned them**, enabling context sensitive logic (if this process called me then do x, otherwise do y).

Claim 3

Claim 3's "The method according to claim 1, wherein the rule datatype includes the rule in an extensible mark-up language file." is anticipated by Rice, et al., column 31, lines 34-58, where it recites:

> Aspects of the virtual machine environment as contemplated by embodiments of the present invention are discussed with regard to FIG. 11. Referring now to FIG. 11, a master agent library 1104 is envisioned to contain those programs that have been previously created and are authorized for use by the current user (e.g., a program such as the one indicated by 1102). It is also envisioned that the master library can be distributed across many computers on a network. A machine-generated agent library 1106 is contemplated to contain portions of the program(s) that are spliced together in a genetic programming environment. Consequently, it is contemplated that these two libraries 1104 and 1106 receive and provide material for the genetic program process. (as well as for other traditional software purposes). Both libraries can store multi-property codon information. In addition the present invention contemplates allowing these two libraries 1104 and 1106 to exist as part of a local machine operating system, and could work with **various file formats** (binary, text, **html, etc.**) In addition, embodiments of the present invention contemplate that machine-generated agent library 1106 can be set up, for example, to set aside a particular area for storing those pieces of code whose score indicates a high level of fitness so that they will not individually be broken apart (although they may continue to be spliced together with other pieces).

Claim 4

Claim 4's "The method according to claim 1, wherein the rule datatype includes a rule identifier of the rule." is anticipated by Rice, et al., column 5, lines 25-35, where it recites:

> The ability to tag and thereby individualize instructions would have many uses. For example, the authorship of code segments could be marked and usage statistics kept. Mutation rates could be assigned to each individual instruction. Unutilized features of a program could be identified and spliced out, while highly used areas could be selected for breeding or identified for software pricing. The inability to individualize each instruction in current environments thus limits the possibility to analyze, study, research, modify, and price programs.

Claim 5

Claim 5's "The method according to claim 1, wherein the rule datatype includes a version of the rule." is anticipated by Rice, et al., column 11, lines 49-63, where it recites:

> Second, the present invention also contemplates that an agent can be written by a user, randomly or systematically generated or evolved, and immediately added to the virtual machine instructions set (e.g., immediately added to the master codon table) using, e.g., a "DefineAgent" function, mentioned below. In this way, the programming language need not be re-compiled. Doing this allows the newly generated items to be used, immediately facilitating real time extension of the operating environment and the ability to map old agents to newly created agents. In addition, it, e.g., enlarges the overall pool from which **succeeding generations** can be bred while performing genetic programming. This concept can also be used to automatically and immediately operate on codons in master codon table 304 that are used in a user-created program.

Claim 6

Claim 6's "The method according to claim 1, wherein the rule datatype includes an input data identifier of the input data used by the rule." is anticipated by Rice, et al., column 9, lines 5-14, where it recites:

> Referring now to FIG. 2, two programs (202 and 204) are shown, each having ten steps. The letters shown in each program represent arguments (e.g., constants) that are used as input to, e.g., subsequent operations (such as an addition operation), represented by the numbers in the program. Each number, in addition to merely signifying the existence of an operation, indicates the number of arguments required by the operation. (Of course, the numbers could represent any type of entity requiring arguments.)

Claim 7

Claim 7's "The method according to claim 1, wherein the rule datatype includes an output identifier of an output of the rule." is anticipated by Rice, et al., column 47, lines 27-29, where it recites:

> DefineAgent — Creates an agent. Takes two arguments, the first is the name of the agent being defined, and the second is the number of arguments the agent takes. **Codon.Label** returns the text label which is used to refer to the codon in input and **output**.

Claim 8

Claim 8's "The method according to claim 1, wherein the output of the rule comprises an indication of a potential exposure to failure of the computer-based system." is anticipated by Rice, et al., column 55, lines 15-30, where it recites:

> To perform self-modification and error handling requires state information ("introspection"). As a design philosophy, one may attempt to make all state information generated by the Virtual Machine in evaluating agents available to the agents themselves. Metaphorically, to correct errors or make adjustments, one needs to know "Where am I? What went wrong? What was the configuration at the time of the error?" Embodiments of the present invention envision that the virtual machine environment consists of an array of state records and some global variables (userCount, steps, errors, lasterror, etc.). All this information is available in read-only mode to the agents. Agents can also query information about the processes

> that spawned them, enabling context sensitive logic (if this process called me then do x, otherwise do y).

Claim 9

Claim 9's "receiving a rule as input from a user," is anticipated by Rice, et al., column 10, lines 38-55, where it recites:

> Referring now to FIG. 3, master codon table 304 is shown containing ten (10) codons (numbered 1 through 10), as indicated by a codon number column 306 in the master codon table 304. The entity that each codon number specifically represents is in the "label" column 308. Thus, codon number 3 represents the "*" operation (i.e., multiplication), codon number 5 represents the constant "9", codon number 10 represents a function called "xlCell.RValue" (which is an abbreviation for Excel read value), etc. (In this example, the Excel program from Microsoft Corporation is mentioned, but any other functionally similar program could also have been used.) The codons shown and described in this Figure were specifically chosen to illustrate the implementation of the above-mentioned temperature conversion program, though it should be clearly understood that the present invention contemplates that the master codon table can contain any number and different types of **codons** (pre-set or **user-defined**).

Claim 9's "the rule defining a logic for determining exposure to failure of a computer-based system based on input data about the computer-based system;" is anticipated by Rice, et al., column 55, lines 15-30, where it recites:

> To perform self-modification and error handling requires state information ("introspection"). As a design philosophy, one may attempt to make all state information generated by the Virtual Machine in evaluating agents available to the agents themselves. Metaphorically, to correct errors or make adjustments, one needs to know "Where am I? What went wrong? What was the configuration at the time of the error?" Embodiments of the present invention envision that the virtual machine environment consists of an array of state records and some global variables (userCount, steps, errors, lasterror, etc.). All this information is available in read-only mode to the agents. Agents can also query information about the processes

> that spawned them, enabling context sensitive logic (if this process called me then do x, otherwise do y).

Claim 9's "preparing a rule datatype including the rule; and" is anticipated by Rice, et al., Abstract, where it recites:

> A fault tolerant software environment, in which various program components (e.g., portions of computer programs, applications, etc) are objectized into entities represented by "codons." This allows for improper syntax to occur, enabling, for example, combinatorial operations such as genetic programming. The present invention also contemplates such features as the ability to probabilistically execute individual codons, to switch between **treating information as executable code or as data** (or passing over it), provides that the individual codons can be tagged so that additional information can be associated with them, and provides for tagging of the stack.

Claim 9's "publishing the rule datatype to a network connected to the data processing system." is anticipated by Rice, et al., column 42, lines 36-52, where it recites:

> Client/Server **codons** allow for many possible **network** architectures, and embodiments are shown with regard to FIG. 20. Referring to FIG. 20, in the example depicted by that Figure, a request to perform a given task is initiated by a client computer X (2002) and sent to Computer A for processing. Using the features envisioned by the present invention, the request could then be routed from computer A to, depending upon load, resources, etc., one or more of computers B, C, D, etc. Assuming (in this example) that Computer D (2010) processes (or helps process) the request, **it is shown that Computer D then passes the result back to client computer X (2002).** In addition to this specific example, FIG. 20 also indicates that more than one client can be involved in a given transaction. It should be understood that the virtual machine environment is contemplated to contain or be contained in, one or more of the various computers/clients shown.

Claim 10

Claim 10's "issuing a query to the network requesting a subscriber identifier of a subscriber to the rule datatype; and" is anticipated by Rice, et al., column 55, lines 16-30, where it recites:

> To perform self-modification and error handling requires state information ("introspection"). As a design philosophy, one may attempt to make all state information generated by the Virtual Machine in evaluating agents available to the agents themselves. Metaphorically, to correct errors or make adjustments, one needs to know "Where am I? What went wrong? What was the configuration at the time of the error?" Embodiments of the present invention envision that the virtual machine environment consists of an array of state records and some global variables (userCount, steps, errors, lasterror, etc.). All this information is available in read-only mode to the agents. Agents can also **query information about the processes that spawned them**, enabling context sensitive logic (if this process called me then do x, otherwise do y).

Claim 10's "receiving the subscriber identifier responsive to the issued query," is anticipated by Rice, et al., column 1, lines 34-52, where it recites:

> The present invention relates to a fault tolerant and combinatorial software environment. More specifically, the present invention relates to a system, method and medium for providing and facilitating a virtual machine and instruction format that facilitates systematic or arbitrary alteration and recombination of portions of code (e.g., programs) at run-time. Instructions that comprise a language that can be used with the present invention can be individualized, allowing for differentiation of each instruction based on, e.g., usage, origin, authorship, or **any other user-defined tag**. The overall instruction set can be extended to include any object or procedure that can be represented by software. The invention is envisioned for use in applications such as those relating to automatic program creation (e.g. genetic programming), artificial/directed evolution of digital designs (graphic, industrial, mechanical, architectural, engineering, etc.), online adaptive software systems, product and process reliability testing, and simulation of biological, non-linear, and probabilistic processes.

Claim 10's "wherein the rule datatype includes the subscriber identifier." is anticipated by Rice, et al., column 1, lines 34-52, where it recites:

> The present invention relates to a fault tolerant and combinatorial software environment. More specifically, the present invention relates to a system, method and medium for providing and facilitating a virtual machine and instruction format that facilitates systematic or arbitrary alteration and recombination of portions of code (e.g., programs) at run-time. Instructions that comprise a language that can be used with the present invention can be individualized, allowing for differentiation of each instruction based on, e.g., usage, origin, authorship, or **any other user-defined tag**. The overall instruction set can be extended to include any object or procedure that can be represented by software. The invention is envisioned for use in applications such as those relating to automatic program creation (e.g. genetic programming), artificial/directed evolution of digital designs (graphic, industrial, mechanical, architectural, engineering, etc.), online adaptive software systems, product and process reliability testing, and simulation of biological, non-linear, and probabilistic processes.

Claim 11

Claim 11's "The computer-readable medium according to claim 9, wherein the rule datatype includes the rule in an extensible mark-up language file." is anticipated by Rice, et al., column 31, lines 34-58, where it recites:

> Aspects of the virtual machine environment as contemplated by embodiments of the present invention are discussed with regard to FIG. 11. Referring now to FIG. 11, a master agent library 1104 is envisioned to contain those programs that have been previously created and are authorized for use by the current user (e.g., a program such as the one indicated by 1102). It is also envisioned that the master library can be distributed across many computers on a network. A machine-generated agent library 1106 is contemplated to contain portions of the program(s) that are spliced together in a genetic programming environment. Consequently, it is contemplated that these two libraries 1104 and 1106 receive and provide material for the genetic program process. (as well as for other traditional software purposes). Both libraries can store multi-property codon information. In addition the present invention contemplates allowing these two libraries 1104 and 1106 to exist as part of a local machine operating system, and could work with **various file**

> **formats** (binary, text, **html, etc.**) In addition, embodiments of the present invention contemplate that machine-generated agent library 1106 can be set up, for example, to set aside a particular area for storing those pieces of code whose score indicates a high level of fitness so that they will not individually be broken apart (although they may continue to be spliced together with other pieces).

Claim 12

Claim 12's "The computer-readable medium according to claim 9, wherein the rule datatype includes a rule identifier of the rule." is anticipated by Rice, et al., column 5, lines 25-35, where it recites:

> The ability to tag and thereby individualize instructions would have many uses. For example, the authorship of code segments could be marked and usage statistics kept. Mutation rates could be assigned to each individual instruction. Unutilized features of a program could be identified and spliced out, while highly used areas could be selected for breeding or identified for software pricing. The inability to individualize each instruction in current environments thus limits the possibility to analyze, study, research, modify, and price programs.

Claim 13

Claim 13's "The computer-readable medium according to claim 9, wherein the rule datatype includes a version of the rule." is anticipated by Rice, et al., column 11, lines 49-63, where it recites:

> Second, the present invention also contemplates that an agent can be written by a user, randomly or systematically generated or evolved, and immediately added to the virtual machine instructions set (e.g., immediately added to the master codon table) using, e.g., a "DefineAgent" function, mentioned below. In this way, the programming language need not be re-compiled. Doing this allows the newly generated items to be used, immediately facilitating real time extension of the operating environment and the ability to map old agents to newly created agents. In addition, it, e.g., enlarges the overall pool from which

> **succeeding generations** can be bred while performing genetic programming. This concept can also be used to automatically and immediately operate on codons in master codon table 304 that are used in a user-created program.

Claim 14

Claim 14's "The computer-readable medium according to claim 9, wherein the rule datatype includes an input data identifier of the input data used by the rule." is anticipated by Rice, et al., column 9, lines 5-14, where it recites:

> Referring now to FIG. 2, two programs (202 and 204) are shown, each having ten steps. The letters shown in each program represent arguments (e.g., constants) that are used as input to, e.g., subsequent operations (such as an addition operation), represented by the numbers in the program. Each number, in addition to merely signifying the existence of an operation, indicates the number of arguments required by the operation. (Of course, the numbers could represent any type of entity requiring arguments.)

Claim 15

Claim 15's "The computer-readable medium according to claim 9, wherein the rule datatype includes an output identifier of an output of the rule." is anticipated by Rice, et al., column 47, lines 27-29, where it recites:

DefineAgent	Creates an agent. Takes two arguments, the first is the name of the agent being defined, and the second is the number of arguments the agent takes. **Codon.Label** returns the text label which is used to refer to the codon in input and **output**.

Claim 16

Claim 16's "The computer-readable medium according to claim 9, wherein the output of the rule comprises an indication of a potential exposure to failure of the computer-based system." is anticipated by Rice, et al., column 55, lines 15-30, where it recites:

> To perform self-modification and error handling requires state information ("introspection"). As a design philosophy, one may attempt to make all state information generated by the Virtual Machine in evaluating agents available to the agents themselves. Metaphorically, to correct errors or make adjustments, one needs to know "Where am I? What went wrong? What was the configuration at the time of the error?" Embodiments of the present invention envision that the virtual machine environment consists of an array of state records and some global variables (userCount, steps, errors, lasterror, etc.). All this information is available in read-only mode to the agents. Agents can also query information about the processes that spawned them, enabling context sensitive logic (if this process called me then do x, otherwise do y).

Claim 17

Claim 17's "a memory comprising a rule publisher program that receives a rule as input from a user," is anticipated by Rice, et al., column 42, lines 36-52, where it recites:

> Client/Server **codons** allow for many possible **network** architectures, and embodiments are shown with regard to FIG. 20. Referring to FIG. 20, in the example depicted by that Figure, a request to perform a given task is initiated by a client computer X (2002) and sent to Computer A for processing. Using the features envisioned by the present invention, the request could then be routed from computer A to, depending upon load, resources, etc., one or more of computers B, C, D, etc. Assuming (in this example) that Computer D (2010) processes (or helps process) the request, **it is shown that Computer D then passes the result back to client computer X (2002).** In addition to this specific example, FIG. 20 also indicates that more than one client can be involved in a given transaction. It should be understood that the virtual machine environment is contemplated to contain or be contained in, one or more of the various computers/clients shown.

Claim 17's "the rule defining a logic for determining exposure to failure of a computer-based system based on input data about the computer-based system," is anticipated by Rice, et al., column 55, lines 15-30, where it recites:

> To perform self-modification and error handling requires state information ("introspection"). As a design philosophy, one may attempt to make all state information generated by the Virtual Machine in evaluating agents available to the agents themselves. Metaphorically, to correct errors or make adjustments, one needs to know "Where am I? What went wrong? What was the configuration at the time of the error?" Embodiments of the present invention envision that the virtual machine environment consists of an array of state records and some global variables (userCount, steps, errors, lasterror, etc.). All this information is available in read-only mode to the agents. Agents can also query information about the processes that spawned them, enabling context sensitive logic (if this process called me then do x, otherwise do y).

Claim 17's "prepares a rule datatype including the rule, and" is anticipated by Rice, et al., Abstract, where it recites:

> A fault tolerant software environment, in which various program components (e.g., portions of computer programs, applications, etc) are objectized into entities represented by "codons." This allows for improper syntax to occur, enabling, for example, combinatorial operations such as genetic programming. The present invention also contemplates such features as the ability to probabilistically execute individual codons, to switch between **treating information as executable code or as data** (or passing over it), provides that the individual codons can be tagged so that additional information can be associated with them, and provides for tagging of the stack.

Claim 17's "publishes the rule datatype to a network connected to the data processing system; and" is anticipated by Rice, et al., column 42, lines 36-52, where it recites:

> Client/Server **codons** allow for many possible **network** architectures, and embodiments are shown with regard to FIG. 20. Referring to FIG. 20, in the example depicted by that Figure, a request to perform a given task is initiated by a client computer X (2002) and sent to Computer A for processing. Using the features envisioned by the present invention, the request could then be routed from computer A to, depending upon load, resources, etc., one or more of computers B, C, D, etc. Assuming (in this example) that Computer D (2010) processes (or helps process) the request, **it is shown that Computer D then passes the result back to client computer X (2002).** In addition to this specific example, FIG. 20 also indicates that more than one client can be involved in a given transaction. It should be understood that the virtual machine environment is contemplated to contain or be contained in, one or more of the various computers/clients shown.

Claim 17's "a processing unit that runs the rule publisher program." is anticipated by Rice, et al., column 42, lines 36-52, where it recites:

> Client/Server **codons** allow for many possible **network** architectures, and embodiments are shown with regard to FIG. 20. Referring to FIG. 20, in the example depicted by that Figure, a request to perform a given task is initiated by a client computer X (2002) and sent to Computer A for processing. Using the features envisioned by the present invention, the request could then be routed from computer A to, depending upon load, resources, etc., one or more of computers B, C, D, etc. Assuming (in this example) that Computer D (2010) processes (or helps process) the request, **it is shown that Computer D then passes the result back to client computer X (2002).** In addition to this specific example, FIG. 20 also indicates that more than one client can be involved in a given transaction. It should be understood that the virtual machine environment is contemplated to contain or be contained in, one or more of the various computers/clients shown.

Claim 18

Claim 18's "The data processing system according to claim 17, wherein the rule deployment program issues a query to the network requesting a subscriber identifier of

a subscriber to the rule datatype, and" is anticipated by Rice, et al., column 1, lines 34-52, where it recites:

> The present invention relates to a fault tolerant and combinatorial software environment. More specifically, the present invention relates to a system, method and medium for providing and facilitating a virtual machine and instruction format that facilitates systematic or arbitrary alteration and recombination of portions of code (e.g., programs) at run-time. Instructions that comprise a language that can be used with the present invention can be individualized, allowing for differentiation of each instruction based on, e.g., usage, origin, authorship, or **any other user-defined tag**. The overall instruction set can be extended to include any object or procedure that can be represented by software. The invention is envisioned for use in applications such as those relating to automatic program creation (e.g. genetic programming), artificial/directed evolution of digital designs (graphic, industrial, mechanical, architectural, engineering, etc.), online adaptive software systems, product and process reliability testing, and simulation of biological, non-linear, and probabilistic processes.

Claim 18's "receives the subscriber identifier responsive to the issued query," is anticipated by Rice, et al., column 1, lines 34-52, where it recites:

> The present invention relates to a fault tolerant and combinatorial software environment. More specifically, the present invention relates to a system, method and medium for providing and facilitating a virtual machine and instruction format that facilitates systematic or arbitrary alteration and recombination of portions of code (e.g., programs) at run-time. Instructions that comprise a language that can be used with the present invention can be individualized, allowing for differentiation of each instruction based on, e.g., usage, origin, authorship, or **any other user-defined tag**. The overall instruction set can be extended to include any object or procedure that can be represented by software. The invention is envisioned for use in applications such as those relating to automatic program creation (e.g. genetic programming), artificial/directed evolution of digital designs (graphic, industrial, mechanical, architectural, engineering, etc.), online adaptive software systems, product and process reliability testing, and simulation of biological, non-linear, and probabilistic processes.

Claim 18's "wherein the rule datatype includes the subscriber identifier." is anticipated by Rice, et al., column 1, lines 34-52, where it recites:

> The present invention relates to a fault tolerant and combinatorial software environment. More specifically, the present invention relates to a system, method and medium for providing and facilitating a virtual machine and instruction format that facilitates systematic or arbitrary alteration and recombination of portions of code (e.g., programs) at run-time. Instructions that comprise a language that can be used with the present invention can be individualized, allowing for differentiation of each instruction based on, e.g., usage, origin, authorship, or **any other user-defined tag**. The overall instruction set can be extended to include any object or procedure that can be represented by software. The invention is envisioned for use in applications such as those relating to automatic program creation (e.g. genetic programming), artificial/directed evolution of digital designs (graphic, industrial, mechanical, architectural, engineering, etc.), online adaptive software systems, product and process reliability testing, and simulation of biological, non-linear, and probabilistic processes.

Claim 19

Claim 19's "The data processing system according to claim 17, wherein the rule datatype includes the rule in an extensible mark-up language file." is anticipated by Rice, et al., column 31, lines 34-58, where it recites:

> Aspects of the virtual machine environment as contemplated by embodiments of the present invention are discussed with regard to FIG. 11. Referring now to FIG. 11, a master agent library 1104 is envisioned to contain those programs that have been previously created and are authorized for use by the current user (e.g., a program such as the one indicated by 1102). It is also envisioned that the master library can be distributed across many computers on a network. A machine-generated agent library 1106 is contemplated to contain portions of the program(s) that are spliced together in a genetic programming environment. Consequently, it is contemplated that these two libraries 1104 and 1106 receive and provide material for the genetic program process. (as well as for other traditional software purposes). Both libraries can store multi-property codon information. In addition the present invention contemplates allowing these two libraries 1104 and 1106 to exist as part of a local machine operating system, and could work with **various file**

> **formats** (binary, text, **html, etc.**) In addition, embodiments of the present invention contemplate that machine-generated agent library 1106 can be set up, for example, to set aside a particular area for storing those pieces of code whose score indicates a high level of fitness so that they will not individually be broken apart (although they may continue to be spliced together with other pieces).

Claim 20

Claim 20's "The data processing system according to claim 17, wherein the rule datatype includes a rule identifier of the rule." is anticipated by Rice, et al., column 5, lines 25-35, where it recites:

> The ability to tag and thereby individualize instructions would have many uses. For example, the authorship of code segments could be marked and usage statistics kept. Mutation rates could be assigned to each individual instruction. Unutilized features of a program could be identified and spliced out, while highly used areas could be selected for breeding or identified for software pricing. The inability to individualize each instruction in current environments thus limits the possibility to analyze, study, research, modify, and price programs.

Claim 21

Claim 21's "The data processing system according to claim 17, wherein the rule datatype includes a version of the rule." is anticipated by Rice, et al., column 11, lines 49-63, where it recites:

> Second, the present invention also contemplates that an agent can be written by a user, randomly or systematically generated or evolved, and immediately added to the virtual machine instructions set (e.g., immediately added to the master codon table) using, e.g., a "DefineAgent" function, mentioned below. In this way, the programming language need not be re-compiled. Doing this allows the newly generated items to be used, immediately facilitating real time extension of the operating environment and the ability to map old agents to newly created agents. In addition, it, e.g., enlarges the overall pool from which

> **succeeding generations** can be bred while performing genetic programming. This concept can also be used to automatically and immediately operate on codons in master codon table 304 that are used in a user-created program.

Claim 22

Claim 22's "The data processing system according to claim 17, wherein the rule datatype includes an input data identifier of the input data used by the rule." is anticipated by Rice, et al., column 9, lines 5-14, where it recites:

> Referring now to FIG. 2, two programs (202 and 204) are shown, each having ten steps. The letters shown in each program represent arguments (e.g., constants) that are used as input to, e.g., subsequent operations (such as an addition operation), represented by the numbers in the program. Each number, in addition to merely signifying the existence of an operation, indicates the number of arguments required by the operation. (Of course, the numbers could represent any type of entity requiring arguments.)

Claim 23

Claim 23's "The data processing system according to claim 17, wherein the rule datatype includes an output identifier of an output of the rule." is anticipated by Rice, et al., column 47, lines 27-29, where it recites:

> DefineAgent — Creates an agent. Takes two arguments, the first is the name of the agent being defined, and the second is the number of arguments the agent takes. **Codon.Label** returns the text label which is used to refer to the codon in input and **output**.

Claim 24

Claim 24's "The data processing system according to claim 17, wherein the output of the rule comprises an indication of a potential exposure to failure of the computer-based system." is anticipated by Rice, et al., column 55, lines 15-30, where it recites:

> To perform self-modification and error handling requires state information ("introspection"). As a design philosophy, one may attempt to make all state information generated by the Virtual Machine in evaluating agents available to the agents themselves. Metaphorically, to correct errors or make adjustments, one needs to know "Where am I? What went wrong? What was the configuration at the time of the error?" Embodiments of the present invention envision that the virtual machine environment consists of an array of state records and some global variables (userCount, steps, errors, lasterror, etc.). All this information is available in read-only mode to the agents. Agents can also query information about the processes that spawned them, enabling context sensitive logic (if this process called me then do x, otherwise do y).

Claim 25

Claim 25's "means for receiving a rule as input from a user," is anticipated by Rice, et al., column 10, lines 38-55, where it recites:

> Referring now to FIG. 3, master codon table 304 is shown containing ten (10) codons (numbered 1 through 10), as indicated by a codon number column 306 in the master codon table 304. The entity that each codon number specifically represents is in the "label" column 308. Thus, codon number 3 represents the "*" operation (i.e., multiplication), codon number 5 represents the constant "9", codon number 10 represents a function called "xlCell.RValue" (which is an abbreviation for Excel read value), etc. (In this example, the Excel program from Microsoft Corporation is mentioned, but any other functionally similar program could also have been used.) The codons shown and described in this Figure were specifically chosen to illustrate the implementation of the above-mentioned temperature conversion program, though it should be clearly understood that the present invention contemplates that the master codon table can contain any number and different types of **codons** (pre-set or **user-defined**).

Claim 25's "the rule defining a logic for determining exposure to failure of a computer-based system based on input data about the computer-based system;" is anticipated by Rice, et al., column 55, lines 15-30, where it recites:

> To perform self-modification and error handling requires state information ("introspection"). As a design philosophy, one may attempt to make all state information generated by the Virtual Machine in evaluating agents available to the agents themselves. Metaphorically, to correct errors or make adjustments, one needs to know "Where am I? What went wrong? What was the configuration at the time of the error?" Embodiments of the present invention envision that the virtual machine environment consists of an array of state records and some global variables (userCount, steps, errors, lasterror, etc.). All this information is available in read-only mode to the agents. Agents can also query information about the processes that spawned them, enabling context sensitive logic (if this process called me then do x, otherwise do y).

Claim 25's "means for preparing a rule datatype including the rule; and" is anticipated by Rice, et al., Abstract, where it recites:

> A fault tolerant software environment, in which various program components (e.g., portions of computer programs, applications, etc) are objectized into entities represented by "codons." This allows for improper syntax to occur, enabling, for example, combinatorial operations such as genetic programming. The present invention also contemplates such features as the ability to probabilistically execute individual codons, to switch between **treating information as executable code or as data** (or passing over it), provides that the individual codons can be tagged so that additional information can be associated with them, and provides for tagging of the stack.

Claim 25's "means for publishing the rule datatype to a network connected to the data processing system." is anticipated by Rice, et al., column 42, lines 36-52, where it recites:

> Client/Server **codons** allow for many possible **network** architectures, and embodiments are shown with regard to FIG. 20. Referring to FIG. 20, in the example depicted by that Figure, a request to perform a given task is initiated by a client computer X (2002) and sent to Computer A for processing. Using the features envisioned by the present invention, the request could then be routed from computer A to, depending upon load, resources, etc., one or more of computers B, C, D, etc. Assuming (in this example) that Computer D (2010) processes (or helps process) the request, **it is shown that Computer D then passes the result back to client computer X (2002).** In addition to this specific example, FIG. 20 also indicates that more than one client can be involved in a given transaction. It should be understood that the virtual machine environment is contemplated to contain or be contained in, one or more of the various computers/clients shown.

Claim 26

Claim 26's "extracting a rule information from a subscribed-to rule datatype," is anticipated by Rice, et al., column 1, lines 34-52, where it recites:

> The present invention relates to a fault tolerant and combinatorial software environment. More specifically, the present invention relates to a system, method and medium for providing and facilitating a virtual machine and instruction format that facilitates systematic or arbitrary alteration and recombination of portions of code (e.g., programs) at run-time. Instructions that comprise a language that can be used with the present invention can be individualized, allowing for differentiation of each instruction based on, e.g., usage, origin, authorship, or **any other user-defined tag**. The overall instruction set can be extended to include any object or procedure that can be represented by software. The invention is envisioned for use in applications such as those relating to automatic program creation (e.g. genetic programming), artificial/directed evolution of digital designs (graphic, industrial, mechanical, architectural, engineering, etc.), online adaptive software systems, product and process reliability testing, and simulation of biological, non-linear, and probabilistic processes.

Claim 26's "wherein the rule information includes a rule that defines a logic for determining exposure to failure of a computer-based system based on input data about

the computer-based system," is anticipated by Rice, et al., column 55, lines 15-30, where it recites:

> To perform self-modification and error handling requires state information ("introspection"). As a design philosophy, one may attempt to make all state information generated by the Virtual Machine in evaluating agents available to the agents themselves. Metaphorically, to correct errors or make adjustments, one needs to know "Where am I? What went wrong? What was the configuration at the time of the error?" Embodiments of the present invention envision that the virtual machine environment consists of an array of state records and some global variables (userCount, steps, errors, lasterror, etc.). All this information is available in read-only mode to the agents. Agents can also query information about the processes that spawned them, enabling context sensitive logic (if this process called me then do x, otherwise do y).

Claim 26's "an identifier of the input data used by the rule, and" is anticipated by Rice, et al., column 9, lines 5-14, where it recites:

> Referring now to FIG. 2, two programs (202 and 204) are shown, each having ten steps. The letters shown in each program represent arguments (e.g., constants) that are used as input to, e.g., subsequent operations (such as an addition operation), represented by the numbers in the program. Each number, in addition to merely signifying the existence of an operation, indicates the number of arguments required by the operation. (Of course, the numbers could represent any type of entity requiring arguments.)

Claim 26's "an identifier of the output data output based on execution of the rule;" is anticipated by Rice, et al., column 42, lines 36-52, where it recites:

> Client/Server **codons** allow for many possible **network** architectures, and embodiments are shown with regard to FIG. 20. Referring to FIG. 20, in the example depicted by that Figure, a request to perform a given task is initiated by a client computer X (2002) and sent to Computer A for processing. Using the features envisioned by the present invention, the request could then be routed from computer A to, depending upon load,

> resources, etc., one or more of computers B, C, D, etc. Assuming (in this example) that Computer D (2010) processes (or helps process) the request, **it is shown that Computer D then passes the result back to client computer X (2002).** In addition to this specific example, FIG. 20 also indicates that more than one client can be involved in a given transaction. It should be understood that the virtual machine environment is contemplated to contain or be contained in, one or more of the various computers/clients shown.

Claim 26's "instantiating a rule engine for executing the rule," is anticipated by Rice, et al., column 1, lines 34-52, where it recites:

> The present invention relates to a fault tolerant and combinatorial software environment. More specifically, the present invention relates to a system, method and medium for providing and facilitating a virtual machine and instruction format that facilitates systematic or arbitrary alteration and recombination of portions of code (e.g., programs) at run-time. Instructions that comprise a language that can be used with the present invention can be individualized, allowing for differentiation of each instruction based on, e.g., usage, origin, authorship, or **any other user-defined tag**. The overall instruction set can be extended to include any object or procedure that can be represented by software. The invention is envisioned for use in applications such as those relating to automatic program creation (e.g. genetic programming), artificial/directed evolution of digital designs (graphic, industrial, mechanical, architectural, engineering, etc.), online adaptive software systems, product and process reliability testing, and simulation of biological, non-linear, and probabilistic processes.

Claim 26's "the rule engine subscribing to the identified input data and outputting the identified output data responsive to completing processing of the rule; and" is anticipated by Rice, et al., column 9, lines 5-14, where it recites:

> Referring now to FIG. 2, two programs (202 and 204) are shown, each having ten steps. The letters shown in each program represent arguments (e.g., constants) that are used as input to, e.g., subsequent operations (such as an addition operation), represented by the numbers

> in the program. Each number, in addition to merely signifying the existence of an operation, indicates the number of arguments required by the operation. (Of course, the numbers could represent any type of entity requiring arguments.)

Claim 26's "deploying the rule engine within a wrapper that encapsulates the rule engine, the wrapper adapted to encapsulate a plurality of rule engines and publish the output data from the rule engine." is anticipated by Rice, et al., column 42, lines 36-52, where it recites:

> Client/Server **codons** allow for many possible **network** architectures, and embodiments are shown with regard to FIG. 20. Referring to FIG. 20, in the example depicted by that Figure, a request to perform a given task is initiated by a client computer X (2002) and sent to Computer A for processing. Using the features envisioned by the present invention, the request could then be routed from computer A to, depending upon load, resources, etc., one or more of computers B, C, D, etc. Assuming (in this example) that Computer D (2010) processes (or helps process) the request, **it is shown that Computer D then passes the result back to client computer X (2002).** In addition to this specific example, FIG. 20 also indicates that more than one client can be involved in a given transaction. It should be understood that the virtual machine environment is contemplated to contain or be contained in, one or more of the various computers/clients shown.

Further, is anticipated by Rice, et al., column 11, lines 64-67 column 12, lines 1-19, where it recites:

> Third, it is also contemplated that one can encapsulate third party compiled code (in real time) into a codon provided that a header file and library file (object file) is available. This involves inspecting the header file, and recording the attributes of the desired "extensions" in a data structure maintained by the virtual machine and loading the function's machine instructions into memory. The function (i.e., a function which is part of the third party code) would be assigned a codon in the typical manner, and given a special type (e.g., "user-defined" codon, as mentioned further below). When that codon is encountered, embodiments of the present invention contemplate that the codon is

> passed to a special **wrapper function** which checks if the appropriate arguments and types are available. If so, the wrapper function extracts the arguments from the virtual machine's data stack, casts them into the appropriate type/form (data types) and places them on the microprocessor's stack. The virtual machine invokes the third party function by transferring control to the memory address where the function was stored previously. The value returned by the function, if any, is converted into the virtual machine's data representation format and placed on its stack similar to any other intrinsic virtual machine function.

Claim 27

Claim 27's "initializing the wrapper for encapsulating the rule engine; and" is anticipated by Rice, et al., column 11, lines 64-67 and column 12, lines 1-19, where it recites:

> Third, it is also contemplated that one can encapsulate third party compiled code (in real time) into a codon provided that a header file and library file (object file) is available. This involves inspecting the header file, and recording the attributes of the desired "extensions" in a data structure maintained by the virtual machine and loading the function's machine instructions into memory. The function (i.e., a function which is part of the third party code) would be assigned a codon in the typical manner, and given a special type (e.g., "user-defined" codon, as mentioned further below). When that codon is encountered, embodiments of the present invention contemplate that the codon is passed to a special **wrapper function** which checks if the appropriate arguments and types are available. If so, the wrapper function extracts the arguments from the virtual machine's data stack, casts them into the appropriate type/form (data types) and places them on the microprocessor's stack. The virtual machine invokes the third party function by transferring control to the memory address where the function was stored previously. The value returned by the function, if any, is converted into the virtual machine's data representation format and placed on its stack similar to any other intrinsic virtual machine function.

Claim 27's "deploying the initialized wrapper." is anticipated by Rice, et al., column 11, lines 64-67 and column 12, lines 1-19, where it recites:

> Third, it is also contemplated that one can encapsulate third party compiled code (in real time) into a codon provided that a header file and library file (object file) is available. This involves inspecting the header file, and recording the attributes of the desired "extensions" in a data structure maintained by the virtual machine and loading the function's machine instructions into memory. The function (i.e., a function which is part of the third party code) would be assigned a codon in the typical manner, and given a special type (e.g., "user-defined" codon, as mentioned further below). When that codon is encountered, embodiments of the present invention contemplate that the codon is passed to a special **wrapper function** which checks if the appropriate arguments and types are available. If so, the wrapper function extracts the arguments from the virtual machine's data stack, casts them into the appropriate type/form (data types) and places them on the microprocessor's stack. The virtual machine invokes the third party function by transferring control to the memory address where the function was stored previously. The value returned by the function, if any, is converted into the virtual machine's data representation format and placed on its stack similar to any other intrinsic virtual machine function.

Claim 28

Claim 28's "receiving the subscribed to rule information." is anticipated by Rice, et al., column 1, lines 34-52, where it recites:

> The present invention relates to a fault tolerant and combinatorial software environment. More specifically, the present invention relates to a system, method and medium for providing and facilitating a virtual machine and instruction format that facilitates systematic or arbitrary alteration and recombination of portions of code (e.g., programs) at run-time. Instructions that comprise a language that can be used with the present invention can be individualized, allowing for differentiation of each instruction based on, e.g., usage, origin, authorship, or **any other user-defined tag**. The overall instruction set can be extended to include any object or procedure that can be represented by software. The invention is envisioned for use in applications such as those relating to automatic program creation (e.g. genetic programming), artificial/directed evolution of digital designs (graphic, industrial, mechanical, architectural, engineering, etc.), online adaptive software systems, product and process reliability testing, and simulation of biological, non-linear, and probabilistic processes.

Claim 29

Claim 29's "receiving at least a second subscribed-to rule datatype;" is anticipated by Rice, et al., column 1, lines 34-52, where it recites:

> The present invention relates to a fault tolerant and combinatorial software environment. More specifically, the present invention relates to a system, method and medium for providing and facilitating a virtual machine and instruction format that facilitates systematic or arbitrary alteration and recombination of portions of code (e.g., programs) at run-time. Instructions that comprise a language that can be used with the present invention can be individualized, allowing for differentiation of each instruction based on, e.g., usage, origin, authorship, or **any other user-defined tag**. The overall instruction set can be extended to include any object or procedure that can be represented by software. The invention is envisioned for use in applications such as those relating to automatic program creation (e.g. genetic programming), artificial/directed evolution of digital designs (graphic, industrial, mechanical, architectural, engineering, etc.), online adaptive software systems, product and process reliability testing, and simulation of biological, non-linear, and probabilistic processes.

Claim 29's "extracting a second rule information from the second subscribed-to rule datatype," is anticipated by Rice, et al., column 1, lines 34-52, where it recites:

> The present invention relates to a fault tolerant and combinatorial software environment. More specifically, the present invention relates to a system, method and medium for providing and facilitating a virtual machine and instruction format that facilitates systematic or arbitrary alteration and recombination of portions of code (e.g., programs) at run-time. Instructions that comprise a language that can be used with the present invention can be individualized, allowing for differentiation of each instruction based on, e.g., usage, origin, authorship, or **any other user-defined tag**. The overall instruction set can be extended to include any object or procedure that can be represented by software. The invention is envisioned for use in applications such as those relating to automatic program creation (e.g. genetic programming), artificial/directed evolution of digital designs (graphic, industrial, mechanical, architectural, engineering, etc.), online adaptive software systems, product and process reliability testing, and simulation of biological, non-linear, and probabilistic processes.

Claim 29's "wherein the second rule information includes a second rule," is anticipated by Rice, et al., column 1, lines 34-52, where it recites:

> The present invention relates to a fault tolerant and combinatorial software environment. More specifically, the present invention relates to a system, method and medium for providing and facilitating a virtual machine and instruction format that facilitates systematic or arbitrary alteration and recombination of portions of code (e.g., programs) at run-time. Instructions that comprise a language that can be used with the present invention can be individualized, allowing for differentiation of each instruction based on, e.g., usage, origin, authorship, or **any other user-defined tag**. The overall instruction set can be extended to include any object or procedure that can be represented by software. The invention is envisioned for use in applications such as those relating to automatic program creation (e.g. genetic programming), artificial/directed evolution of digital designs (graphic, industrial, mechanical, architectural, engineering, etc.), online adaptive software systems, product and process reliability testing, and simulation of biological, non-linear, and probabilistic processes.

Claim 29's "an identifier of the input data used by the second rule, and" is anticipated by Rice, et al., column 1, lines 34-52, where it recites:

> The present invention relates to a fault tolerant and combinatorial software environment. More specifically, the present invention relates to a system, method and medium for providing and facilitating a virtual machine and instruction format that facilitates systematic or arbitrary alteration and recombination of portions of code (e.g., programs) at run-time. Instructions that comprise a language that can be used with the present invention can be individualized, allowing for differentiation of each instruction based on, e.g., usage, origin, authorship, or **any other user-defined tag**. The overall instruction set can be extended to include any object or procedure that can be represented by software. The invention is envisioned for use in applications such as those relating to automatic program creation (e.g. genetic programming), artificial/directed evolution of digital designs (graphic, industrial, mechanical, architectural, engineering, etc.), online adaptive software systems, product and process reliability testing, and simulation of biological, non-linear, and probabilistic processes.

Claim 29's "an identifier of the output data output based on execution of the second rule;" is anticipated by Rice, et al., column 1, lines 34-52, where it recites:

> The present invention relates to a fault tolerant and combinatorial software environment. More specifically, the present invention relates to a system, method and medium for providing and facilitating a virtual machine and instruction format that facilitates systematic or arbitrary alteration and recombination of portions of code (e.g., programs) at run-time. Instructions that comprise a language that can be used with the present invention can be individualized, allowing for differentiation of each instruction based on, e.g., usage, origin, authorship, or **any other user-defined tag**. The overall instruction set can be extended to include any object or procedure that can be represented by software. The invention is envisioned for use in applications such as those relating to automatic program creation (e.g. genetic programming), artificial/directed evolution of digital designs (graphic, industrial, mechanical, architectural, engineering, etc.), online adaptive software systems, product and process reliability testing, and simulation of biological, non-linear, and probabilistic processes.

Claim 29's "instantiating a second rule engine for executing the second rule; and" is anticipated by Rice, et al., column 1, lines 34-52, where it recites:

> The present invention relates to a fault tolerant and combinatorial software environment. More specifically, the present invention relates to a system, method and medium for providing and facilitating a virtual machine and instruction format that facilitates systematic or arbitrary alteration and recombination of portions of code (e.g., programs) at run-time. Instructions that comprise a language that can be used with the present invention can be individualized, allowing for differentiation of each instruction based on, e.g., usage, origin, authorship, or **any other user-defined tag**. The overall instruction set can be extended to include any object or procedure that can be represented by software. The invention is envisioned for use in applications such as those relating to automatic program creation (e.g. genetic programming), artificial/directed evolution of digital designs (graphic, industrial, mechanical, architectural, engineering, etc.), online adaptive software systems, product and process reliability testing, and simulation of biological, non-linear, and probabilistic processes.

Claim 29's "deploying the second rule engine within at least one of the wrapper, which encapsulates the rule engine, and a different wrapper." is anticipated by Rice, et al., column 11, lines 64-67 and column 12, lines 1-19, where it recites:

> Third, it is also contemplated that one can encapsulate third party compiled code (in real time) into a codon provided that a header file and library file (object file) is available. This involves inspecting the header file, and recording the attributes of the desired "extensions" in a data structure maintained by the virtual machine and loading the function's machine instructions into memory. The function (i.e., a function which is part of the third party code) would be assigned a codon in the typical manner, and given a special type (e.g., "user-defined" codon, as mentioned further below). When that codon is encountered, embodiments of the present invention contemplate that the codon is passed to a special **wrapper function** which checks if the appropriate arguments and types are available. If so, the wrapper function extracts the arguments from the virtual machine's data stack, casts them into the appropriate type/form (data types) and places them on the microprocessor's stack. The virtual machine invokes the third party function by transferring control to the memory address where the function was stored previously. The value returned by the function, if any, is converted into the virtual machine's data representation format and placed on its stack similar to any other intrinsic virtual machine function.

Claim 30

Claim 30's "The method according to claim 26, wherein the rule information is within an extensible mark-up language file." is anticipated by Rice, et al., column 31, lines 34-58, where it recites:

> Aspects of the virtual machine environment as contemplated by embodiments of the present invention are discussed with regard to FIG. 11. Referring now to FIG. 11, a master agent library 1104 is envisioned to contain those programs that have been previously created and are authorized for use by the current user (e.g., a program such as the one indicated by 1102). It is also envisioned that the master library can be distributed across many computers on a network. A machine-generated agent library 1106 is contemplated to contain portions of the program(s) that are spliced together in a genetic programming environment.

> Consequently, it is contemplated that these two libraries 1104 and 1106 receive and provide material for the genetic program process. (as well as for other traditional software purposes). Both libraries can store multi-property codon information. In addition the present invention contemplates allowing these two libraries 1104 and 1106 to exist as part of a local machine operating system, and could work with **various file formats** (binary, text, **html, etc.**) In addition, embodiments of the present invention contemplate that machine-generated agent library 1106 can be set up, for example, to set aside a particular area for storing those pieces of code whose score indicates a high level of fitness so that they will not individually be broken apart (although they may continue to be spliced together with other pieces).

Claim 31

Claim 31's "The method according to claim 26, wherein the rule information includes a preliminary rule that is deployed in the wrapper with the rule, the preliminary rule being executed by the rule engine prior to executing the rule." is anticipated by Rice, et al., column 11, lines 64-67 and column 12, lines 1-19, where it recites:

> Third, it is also contemplated that one can encapsulate third party compiled code (in real time) into a codon provided that a header file and library file (object file) is available. This involves inspecting the header file, and recording the attributes of the desired "extensions" in a data structure maintained by the virtual machine and loading the function's machine instructions into memory. The function (i.e., a function which is part of the third party code) would be assigned a codon in the typical manner, and given a special type (e.g., "user-defined" codon, as mentioned further below). When that codon is encountered, embodiments of the present invention contemplate that the codon is passed to a special **wrapper function** which checks if the appropriate arguments and types are available. If so, the wrapper function extracts the arguments from the virtual machine's data stack, casts them into the appropriate type/form (data types) and places them on the microprocessor's stack. The virtual machine invokes the third party function by transferring control to the memory address where the function was stored previously. The value returned by the function, if any, is converted into the virtual machine's data representation format and placed on its stack similar to any other intrinsic virtual machine function.

Claim 32

Claim 32's "The method according to claim 26, wherein the output data from the rule engine is subscribed to by another rule engine." is anticipated by Rice, et al., column 42, lines 36-52, where it recites:

> Client/Server **codons** allow for many possible **network** architectures, and embodiments are shown with regard to FIG. 20. Referring to FIG. 20, in the example depicted by that Figure, a request to perform a given task is initiated by a client computer X (2002) and sent to Computer A for processing. Using the features envisioned by the present invention, the request could then be routed from computer A to, depending upon load, resources, etc., one or more of computers B; C, D, etc. Assuming (in this example) that Computer D (2010) processes (or helps process) the request, **it is shown that Computer D then passes the result back to client computer X (2002).** In addition to this specific example, FIG. 20 also indicates that more than one client can be involved in a given transaction. It should be understood that the virtual machine environment is contemplated to contain or be contained in, one or more of the various computers/clients shown.

Claim 33

Claim 33's "The method according to claim 26, wherein the output data includes an indication of a potential exposure to failure of the computer-based system." is anticipated by Rice, et al., column 55, lines 15-30, where it recites:

> To perform self-modification and error handling requires state information ("introspection"). As a design philosophy, one may attempt to make all state information generated by the Virtual Machine in evaluating agents available to the agents themselves. Metaphorically, to correct errors or make adjustments, one needs to know "Where am I? What went wrong? What was the configuration at the time of the error?" Embodiments of the present invention envision that the virtual machine environment consists of an array of state records and some global variables (userCount, steps, errors, lasterror, etc.). All this information is available in read-only mode to the agents. Agents can also query information about the processes that spawned them, enabling context sensitive logic (if this process called me then do x, otherwise do y).

Claim 34

Claim 34's "The method according to claim 26, wherein the output data includes an exposure level to failure of the computer-based system." is anticipated by Rice, et al., column 42, lines 36-52, where it recites:

> Client/Server **codons** allow for many possible **network** architectures, and embodiments are shown with regard to FIG. 20. Referring to FIG. 20, in the example depicted by that Figure, a request to perform a given task is initiated by a client computer X (2002) and sent to Computer A for processing. Using the features envisioned by the present invention, the request could then be routed from computer A to, depending upon load, resources, etc., one or more of computers B, C, D, etc. Assuming (in this example) that Computer D (2010) processes (or helps process) the request, **it is shown that Computer D then passes the result back to client computer X (2002).** In addition to this specific example, FIG. 20 also indicates that more than one client can be involved in a given transaction. It should be understood that the virtual machine environment is contemplated to contain or be contained in, one or more of the various computers/clients shown.

Claim 35

Claim 35's "The method according to claim 34, wherein the output data includes a confidence level of the exposure level." is anticipated by Rice, et al., column 29, lines 35-55, where it recites:

> Still referring to program 902, Step 2 is the cosine operation which, requiring one argument as input, receives the 11 from stack 904, and yields 0.9816, as shown at Step 3 of stack 904. Then, upon encountering Step 3 of the program 902, which is the multiplication operation, the present invention looks for two arguments. Finding only one (i.e., the 0.9816), the present invention's grammar repair generates a second argument (which in this example is the number 3) and places it on the stack at step 3 as shown. The multiplication can then commence, yielding the number 2.9448. Thus, the program 902 can then be tested for **fitness** using the numbers generated, as shown on stack 904. It may very well be that the results are unlikely to score well with the fitness function, but at least there will be some result to work with. Embodiments

> of the current invention contemplate that the specific type of grammar repair function used can differ depending upon the specific codon(s) encountered. Furthermore, it is envisioned that the grammar repair function can vary with each codon and can be set at any time.

Claim 36

Claim 36's "The method according to claim 26, wherein the output data identifies whether the rule engine completed execution of the rule." is anticipated by Rice, et al., column 42, lines 36-52, where it recites:

> Client/Server **codons** allow for many possible **network** architectures, and embodiments are shown with regard to FIG. 20. Referring to FIG. 20, in the example depicted by that Figure, a request to perform a given task is initiated by a client computer X (2002) and sent to Computer A for processing. Using the features envisioned by the present invention, the request could then be routed from computer A to, depending upon load, resources, etc., one or more of computers B, C, D, etc. Assuming (in this example) that Computer D (2010) processes (or helps process) the request, **it is shown that Computer D then passes the result back to client computer X (2002).** In addition to this specific example, FIG. 20 also indicates that more than one client can be involved in a given transaction. It should be understood that the virtual machine environment is contemplated to contain or be contained in, one or more of the various computers/clients shown.

Claim 37

Claim 37's "The method according to claim 26, wherein the output data identifies that an error occurred during execution of the rule." is anticipated by Rice, et al., column 13, lines 52-67 and column 14, lines 1-11, where it recites:

> "T.sub.-- DATA.sub.-- INDEX" is an implementation-defined integral data type such as "long" whose value is set to the number of items in the return buffer or to a negative number to **indicate that an error occurred**. For example, a "-1" could indicate that an error occurred (usually due to invalid arguments) and the return buffer would be empty.

> It is further envisioned that there would exist a table of errors which would contain error numbers and error descriptions which could be extended during runtime. A return value of "0" could indicate that the virtual machine function or procedure was successfully executed, but that no values were returned (i.e. the return buffer is empty). Any integer greater than "0" could indicate that the function was successfully executed and the integer returned would indicate the number of items in the return buffer. The value and type of the items could then be accessed in the return buffer. While embodiments of the present invention envision that the argument data types can be determined by the evaluating functions by passing arguments to and from the virtual machines in the form of variants and providing mechanisms for retrieving and converting fundamental and structure types into variants, other possible schemes are also contemplated, including maintaining a table in the virtual machine of the type of arguments each function takes. In the former case, type checking is carried out by the virtual machine functions, in the latter case, the virtual machine itself determines whether the appropriate arguments are available.

Claim 38

Claim 38's "The method according to claim 26, wherein the output data identifies that a side effect occurred during execution of the rule." is anticipated by Rice, et al., column 13, lines 52-67 and column 14, lines 1-11, where it recites:

> "T.sub.-- DATA.sub.-- INDEX" is an implementation-defined integral data type such as "long" whose value is set to the number of items in the return buffer or to a negative number to **indicate that an error occurred**. For example, a "-1" could indicate that an error occurred (usually due to invalid arguments) and the return buffer would be empty. It is further envisioned that there would exist a table of errors which would contain error numbers and error descriptions which could be extended during runtime. A return value of "0" could indicate that the virtual machine function or procedure was successfully executed, but that no values were returned (i.e. the return buffer is empty). Any integer greater than "0" could indicate that the function was successfully executed and the integer returned would indicate the number of items in the return buffer. The value and type of the items could then be accessed in the return buffer. While embodiments of the present invention envision that the argument data types can be determined by the evaluating functions by passing arguments to and from the virtual machines in the form of variants and providing mechanisms for retrieving and converting fundamental and structure types into variants, other possible schemes are also contemplated, including maintaining a table in the virtual machine of the type of arguments each function takes. In the

> former case, type checking is carried out by the virtual machine functions, in the latter case, the virtual machine itself determines whether the appropriate arguments are available.

Claim 39

Claim 39's "extracting a rule information from a subscribed-to rule datatype," is anticipated by Rice, et al., column 1, lines 34-52, where it recites:

> The present invention relates to a fault tolerant and combinatorial software environment. More specifically, the present invention relates to a system, method and medium for providing and facilitating a virtual machine and instruction format that facilitates systematic or arbitrary alteration and recombination of portions of code (e.g., programs) at run-time. Instructions that comprise a language that can be used with the present invention can be individualized, allowing for differentiation of each instruction based on, e.g., usage, origin, authorship, or **any other user-defined tag**. The overall instruction set can be extended to include any object or procedure that can be represented by software. The invention is envisioned for use in applications such as those relating to automatic program creation (e.g. genetic programming), artificial/directed evolution of digital designs (graphic, industrial, mechanical, architectural, engineering, etc.), online adaptive software systems, product and process reliability testing, and simulation of biological, non-linear, and probabilistic processes.

Claim 39's "wherein the rule information includes a rule that defines a logic for determining exposure to failure of a computer-based system based on input data about the computer-based system," is anticipated by Rice, et al., column 55, lines 15-30, where it recites:

> To perform self-modification and error handling requires state information ("introspection"). As a design philosophy, one may attempt to make all state information generated by the Virtual Machine in evaluating agents available to the agents themselves. Metaphorically, to correct errors or make adjustments, one needs to know "Where am I? What went wrong? What was the configuration at the time of the error?" Embodiments of the present invention envision that the virtual machine environment consists

> of an array of state records and some global variables (userCount, steps, errors, lasterror, etc.). All this information is available in read-only mode to the agents. Agents can also query information about the processes that spawned them, enabling context sensitive logic (if this process called me then do x, otherwise do y).

Claim 39's "an identifier of the input data used by the rule, and" is anticipated by Rice, et al., column 9, lines 5-14, where it recites:

> Referring now to FIG. 2, two programs (202 and 204) are shown, each having ten steps. The letters shown in each program represent arguments (e.g., constants) that are used as input to, e.g., subsequent operations (such as an addition operation), represented by the numbers in the program. Each number, in addition to merely signifying the existence of an operation, indicates the number of arguments required by the operation. (Of course, the numbers could represent any type of entity requiring arguments.)

Claim 39's "an identifier of the output data output based on execution of the rule;" is anticipated by Rice, et al., column 42, lines 36-52, where it recites:

> Client/Server **codons** allow for many possible **network** architectures, and embodiments are shown with regard to FIG. 20. Referring to FIG. 20, in the example depicted by that Figure, a request to perform a given task is initiated by a client computer X (2002) and sent to Computer A for processing. Using the features envisioned by the present invention, the request could then be routed from computer A to, depending upon load, resources, etc., one or more of computers B, C, D, etc. Assuming (in this example) that Computer D (2010) processes (or helps process) the request, **it is shown that Computer D then passes the result back to client computer X (2002).** In addition to this specific example, FIG. 20 also indicates that more than one client can be involved in a given transaction. It should be understood that the virtual machine environment is contemplated to contain or be contained in, one or more of the various computers/clients shown.

Claim 39's "instantiating a rule engine for executing the rule," is anticipated by Rice, et al., column 1, lines 34-52, where it recites:

> The present invention relates to a fault tolerant and combinatorial software environment. More specifically, the present invention relates to a system, method and medium for providing and facilitating a virtual machine and instruction format that facilitates systematic or arbitrary alteration and recombination of portions of code (e.g., programs) at run-time. Instructions that comprise a language that can be used with the present invention can be individualized, allowing for differentiation of each instruction based on, e.g., usage, origin, authorship, or **any other user-defined tag**. The overall instruction set can be extended to include any object or procedure that can be represented by software. The invention is envisioned for use in applications such as those relating to automatic program creation (e.g. genetic programming), artificial/directed evolution of digital designs (graphic, industrial, mechanical, architectural, engineering, etc.), online adaptive software systems, product and process reliability testing, and simulation of biological, non-linear, and probabilistic processes.

Claim 39's "the rule engine subscribing to the identified input data and outputting the identified output data responsive to completing processing of the rule; and" is anticipated by Rice, et al., column 9, lines 5-14, where it recites:

> Referring now to FIG. 2, two programs (202 and 204) are shown, each having ten steps. The letters shown in each program represent arguments (e.g., constants) that are used as input to, e.g., subsequent operations (such as an addition operation), represented by the numbers in the program. Each number, in addition to merely signifying the existence of an operation, indicates the number of arguments required by the operation. (Of course, the numbers could represent any type of entity requiring arguments.)

Claim 39's "deploying the rule engine within a wrapper that encapsulates the rule engine, the wrapper adapted to encapsulate a plurality of rule engines and publish the output data from the rule engine." is anticipated by Rice, et al., column 42, lines 36-52, where it recites:

> Client/Server **codons** allow for many possible **network** architectures, and embodiments are shown with regard to FIG. 20. Referring to FIG. 20, in the example depicted by that Figure, a request to perform a given task is initiated by a client computer X (2002) and sent to Computer A for processing. Using the features envisioned by the present invention, the request could then be routed from computer A to, depending upon load, resources, etc., one or more of computers B, C, D, etc. Assuming (in this example) that Computer D (2010) processes (or helps process) the request, **it is shown that Computer D then passes the result back to client computer X (2002).** In addition to this specific example, FIG. 20 also indicates that more than one client can be involved in a given transaction. It should be understood that the virtual machine environment is contemplated to contain or be contained in, one or more of the various computers/clients shown.

Further, is anticipated by Rice, et al., column 11, lines 64-67 and column 12, lines 1-19, where it recites:

> Third, it is also contemplated that one can encapsulate third party compiled code (in real time) into a codon provided that a header file and library file (object file) is available. This involves inspecting the header file, and recording the attributes of the desired "extensions" in a data structure maintained by the virtual machine and loading the function's machine instructions into memory. The function (i.e., a function which is part of the third party code) would be assigned a codon in the typical manner, and given a special type (e.g., "user-defined" codon, as mentioned further below). When that codon is encountered, embodiments of the present invention contemplate that the codon is passed to a special **wrapper function** which checks if the appropriate arguments and types are available. If so, the wrapper function extracts the arguments from the virtual machine's data stack, casts them into the appropriate type/form (data types) and places them on the microprocessor's stack. The virtual machine invokes the third party function by transferring control to the memory address where the function was stored previously. The value returned by the function, if any, is converted into the virtual machine's data representation format and placed on its stack similar to any other intrinsic virtual machine function.

Claim 40

Claim 40's "initializing the wrapper for encapsulating the rule engine; and" is anticipated by Rice, et al., column 11, lines 64-67 and column 12, lines 1-19, where it recites:

> Third, it is also contemplated that one can encapsulate third party compiled code (in real time) into a codon provided that a header file and library file (object file) is available. This involves inspecting the header file, and recording the attributes of the desired "extensions" in a data structure maintained by the virtual machine and loading the function's machine instructions into memory. The function (i.e., a function which is part of the third party code) would be assigned a codon in the typical manner, and given a special type (e.g., "user-defined" codon, as mentioned further below). When that codon is encountered, embodiments of the present invention contemplate that the codon is passed to a special **wrapper function** which checks if the appropriate arguments and types are available. If so, the wrapper function extracts the arguments from the virtual machine's data stack, casts them into the appropriate type/form (data types) and places them on the microprocessor's stack. The virtual machine invokes the third party function by transferring control to the memory address where the function was stored previously. The value returned by the function, if any, is converted into the virtual machine's data representation format and placed on its stack similar to any other intrinsic virtual machine function.

Claim 40's "deploying the initialized wrapper." is anticipated by Rice, et al., column 11, lines 64-67 and column 12, lines 1-19, where it recites:

> Third, it is also contemplated that one can encapsulate third party compiled code (in real time) into a codon provided that a header file and library file (object file) is available. This involves inspecting the header file, and recording the attributes of the desired "extensions" in a data structure maintained by the virtual machine and loading the function's machine instructions into memory. The function (i.e., a function which is part of the third party code) would be assigned a codon in the typical manner, and given a special type (e.g., "user-defined" codon, as mentioned further below). When that codon is encountered, embodiments of the present invention contemplate that the codon is passed to a special **wrapper function** which checks if the appropriate arguments and types are available. If so, the wrapper function extracts the arguments from the virtual machine's data stack, casts them into the appropriate type/form (data types) and places them on the

> microprocessor's stack. The virtual machine invokes the third party function by transferring control to the memory address where the function was stored previously. The value returned by the function, if any, is converted into the virtual machine's data representation format and placed on its stack similar to any other intrinsic virtual machine function.

Claim 41

Claim 41's "The computer-readable medium according to claim 39, further comprising the step of: receiving the subscribed to rule information." is anticipated by Rice, et al., column 1, lines 34-52, where it recites:

> The present invention relates to a fault tolerant and combinatorial software environment. More specifically, the present invention relates to a system, method and medium for providing and facilitating a virtual machine and instruction format that facilitates systematic or arbitrary alteration and recombination of portions of code (e.g., programs) at run-time. Instructions that comprise a language that can be used with the present invention can be individualized, allowing for differentiation of each instruction based on, e.g., usage, origin, authorship, or **any other user-defined tag**. The overall instruction set can be extended to include any object or procedure that can be represented by software. The invention is envisioned for use in applications such as those relating to automatic program creation (e.g. genetic programming), artificial/directed evolution of digital designs (graphic, industrial, mechanical, architectural, engineering, etc.), online adaptive software systems, product and process reliability testing, and simulation of biological, non-linear, and probabilistic processes.

Claim 42

Claim 42's "receiving at least a second subscribed-to rule datatype;" is anticipated by Rice, et al., column 1, lines 34-52, where it recites:

> The present invention relates to a fault tolerant and combinatorial software environment. More specifically, the present invention relates to a system, method and medium for providing and facilitating a virtual machine and instruction format that facilitates systematic or arbitrary alteration and recombination of portions of code (e.g., programs) at run-

> time. Instructions that comprise a language that can be used with the present invention can be individualized, allowing for differentiation of each instruction based on, e.g., usage, origin, authorship, or **any other user-defined tag**. The overall instruction set can be extended to include any object or procedure that can be represented by software. The invention is envisioned for use in applications such as those relating to automatic program creation (e.g. genetic programming), artificial/directed evolution of digital designs (graphic, industrial, mechanical, architectural, engineering, etc.), online adaptive software systems, product and process reliability testing, and simulation of biological, non-linear, and probabilistic processes.

Claim 42's "extracting a second rule information from the second subscribed-to rule datatype," is anticipated by Rice, et al., column 1, lines 34-52, where it recites:

> The present invention relates to a fault tolerant and combinatorial software environment. More specifically, the present invention relates to a system, method and medium for providing and facilitating a virtual machine and instruction format that facilitates systematic or arbitrary alteration and recombination of portions of code (e.g., programs) at run-time. Instructions that comprise a language that can be used with the present invention can be individualized, allowing for differentiation of each instruction based on, e.g., usage, origin, authorship, or **any other user-defined tag**. The overall instruction set can be extended to include any object or procedure that can be represented by software. The invention is envisioned for use in applications such as those relating to automatic program creation (e.g. genetic programming), artificial/directed evolution of digital designs (graphic, industrial, mechanical, architectural, engineering, etc.), online adaptive software systems, product and process reliability testing, and simulation of biological, non-linear, and probabilistic processes.

Claim 42's "wherein the second rule information includes a second rule," is anticipated by Rice, et al., column 1, lines 34-52, where it recites:

> The present invention relates to a fault tolerant and combinatorial software environment. More specifically, the present invention relates to a system, method and medium for providing and facilitating a virtual machine and instruction format that facilitates systematic or arbitrary alteration and recombination of portions of code (e.g., programs) at run-time. Instructions that comprise a language that can be used with the

> present invention can be individualized, allowing for differentiation of each instruction based on, e.g., usage, origin, authorship, or **any other user-defined tag**. The overall instruction set can be extended to include any object or procedure that can be represented by software. The invention is envisioned for use in applications such as those relating to automatic program creation (e.g. genetic programming), artificial/directed evolution of digital designs (graphic, industrial, mechanical, architectural, engineering, etc.), online adaptive software systems, product and process reliability testing, and simulation of biological, non-linear, and probabilistic processes.

Claim 42's "an identifier of the input data used by the second rule, and" is anticipated by Rice, et al., column 9, lines 5-14, where it recites:

> Referring now to FIG. 2, two programs (202 and 204) are shown, each having ten steps. The letters shown in each program represent arguments (e.g., constants) that are used as input to, e.g., subsequent operations (such as an addition operation), represented by the numbers in the program. Each number, in addition to merely signifying the existence of an operation, indicates the number of arguments required by the operation. (Of course, the numbers could represent any type of entity requiring arguments.)

Claim 42's "an identifier of the output data output based on execution of the second rule;" is anticipated by Rice, et al., column 42, lines 36-52, where it recites:

> Client/Server **codons** allow for many possible **network** architectures, and embodiments are shown with regard to FIG. 20. Referring to FIG. 20, in the example depicted by that Figure, a request to perform a given task is initiated by a client computer X (2002) and sent to Computer A for processing. Using the features envisioned by the present invention, the request could then be routed from computer A to, depending upon load, resources, etc., one or more of computers B, C, D, etc. Assuming (in this example) that Computer D (2010) processes (or helps process) the request, **it is shown that Computer D then passes the result back to client computer X (2002).** In addition to this specific example, FIG. 20 also indicates that more than one client can be involved in a given transaction. It should be understood that the virtual machine environment is contemplated to contain or be contained in, one or more of the various computers/clients shown.

Claim 42's "instantiating a second rule engine for executing the second rule; and" is anticipated by Rice, et al., column 1, lines 34-52, where it recites:

> The present invention relates to a fault tolerant and combinatorial software environment. More specifically, the present invention relates to a system, method and medium for providing and facilitating a virtual machine and instruction format that facilitates systematic or arbitrary alteration and recombination of portions of code (e.g., programs) at run-time. Instructions that comprise a language that can be used with the present invention can be individualized, allowing for differentiation of each instruction based on, e.g., usage, origin, authorship, or **any other user-defined tag**. The overall instruction set can be extended to include any object or procedure that can be represented by software. The invention is envisioned for use in applications such as those relating to automatic program creation (e.g. genetic programming), artificial/directed evolution of digital designs (graphic, industrial, mechanical, architectural, engineering, etc.), online adaptive software systems, product and process reliability testing, and simulation of biological, non-linear, and probabilistic processes.

Claim 42's "deploying the second rule engine within at least one of the wrapper, which encapsulates the rule engine, and a different wrapper." is anticipated by Rice, et al., column 11, lines 64-67 and column 12, lines 1-19, where it recites:

> Third, it is also contemplated that one can encapsulate third party compiled code (in real time) into a codon provided that a header file and library file (object file) is available. This involves inspecting the header file, and recording the attributes of the desired "extensions" in a data structure maintained by the virtual machine and loading the function's machine instructions into memory. The function (i.e., a function which is part of the third party code) would be assigned a codon in the typical manner, and given a special type (e.g., "user-defined" codon, as mentioned further below). When that codon is encountered, embodiments of the present invention contemplate that the codon is passed to a special **wrapper function** which checks if the appropriate arguments and types are available. If so, the wrapper function extracts the arguments from the virtual machine's data stack, casts them into the appropriate type/form (data types) and places them on the microprocessor's stack. The virtual machine invokes the third party function by transferring control to the memory address where the function was stored previously. The value returned by the function, if any,

> is converted into the virtual machine's data representation format and placed on its stack similar to any other intrinsic virtual machine function.

Claim 43

Claim 43's "The computer-readable medium according to claim 39, wherein the rule information is within an extensible mark-up language file." is anticipated by Rice, et al., column 31, lines 34-58, where it recites:

> Aspects of the virtual machine environment as contemplated by embodiments of the present invention are discussed with regard to FIG. 11. Referring now to FIG. 11, a master agent library 1104 is envisioned to contain those programs that have been previously created and are authorized for use by the current user (e.g., a program such as the one indicated by 1102). It is also envisioned that the master library can be distributed across many computers on a network. A machine-generated agent library 1106 is contemplated to contain portions of the program(s) that are spliced together in a genetic programming environment. Consequently, it is contemplated that these two libraries 1104 and 1106 receive and provide material for the genetic program process. (as well as for other traditional software purposes). Both libraries can store multi-property codon information. In addition the present invention contemplates allowing these two libraries 1104 and 1106 to exist as part of a local machine operating system, and could work with **various file formats** (binary, text, **html, etc.**) In addition, embodiments of the present invention contemplate that machine-generated agent library 1106 can be set up, for example, to set aside a particular area for storing those pieces of code whose score indicates a high level of fitness so that they will not individually be broken apart (although they may continue to be spliced together with other pieces).

Claim 44

Claim 44's "The computer-readable medium according to claim 39, wherein the rule information includes a preliminary rule that is deployed in the wrapper with the rule, " is anticipated by Rice, et al., column 11, lines 64-67 and column 12, lines 1-19, where it recites:

> Third, it is also contemplated that one can encapsulate third party compiled code (in real time) into a codon provided that a header file and library file (object file) is available. This involves inspecting the header file, and recording the attributes of the desired "extensions" in a data structure maintained by the virtual machine and loading the function's machine instructions into memory. The function (i.e., a function which is part of the third party code) would be assigned a codon in the typical manner, and given a special type (e.g., "user-defined" codon, as mentioned further below). When that codon is encountered, embodiments of the present invention contemplate that the codon is passed to a special **wrapper function** which checks if the appropriate arguments and types are available. If so, the wrapper function extracts the arguments from the virtual machine's data stack, casts them into the appropriate type/form (data types) and places them on the microprocessor's stack. The virtual machine invokes the third party function by transferring control to the memory address where the function was stored previously. The value returned by the function, if any, is converted into the virtual machine's data representation format and placed on its stack similar to any other intrinsic virtual machine function.

Claim 44's "the preliminary rule being executed by the rule engine prior to executing the rule." is anticipated by Rice, et al., column 11, lines 64-67 and column 12, lines 1-19, where it recites:

> Third, it is also contemplated that one can encapsulate third party compiled code (in real time) into a codon provided that a header file and library file (object file) is available. This involves inspecting the header file, and recording the attributes of the desired "extensions" in a data structure maintained by the virtual machine and loading the function's machine instructions into memory. The function (i.e., a function which is part of the third party code) would be assigned a codon in the typical manner, and given a special type (e.g., "user-defined" codon, as mentioned further below). When that codon is encountered, embodiments of the present invention contemplate that the codon is passed to a special **wrapper function** which **checks if the appropriate arguments and types are available**. If so, the wrapper function extracts the arguments from the virtual machine's data stack, **casts** them into the appropriate type/form (data types) and **places** them on the microprocessor's stack. The virtual machine invokes the third party function by transferring control to the memory address where the function was stored previously. The value returned by the function, if any, is converted into the virtual machine's data representation format and placed on its stack similar to any other intrinsic virtual machine function.

Claim 45

Claim 45's "The computer-readable medium according to claim 39, wherein the output data from the rule engine is subscribed to by another rule engine." is anticipated by Rice, et al., column 42, lines 36-52, where it recites:

> Client/Server **codons** allow for many possible **network** architectures, and embodiments are shown with regard to FIG. 20. Referring to FIG. 20, in the example depicted by that Figure, a request to perform a given task is initiated by a client computer X (2002) and sent to Computer A for processing. Using the features envisioned by the present invention, the request could then be routed from computer A to, depending upon load, resources, etc., one or more of computers B, C, D, etc. Assuming (in this example) that Computer D (2010) processes (or helps process) the request, **it is shown that Computer D then passes the result back to client computer X (2002).** In addition to this specific example, FIG. 20 also indicates that more than one client can be involved in a given transaction. It should be understood that the virtual machine environment is contemplated to contain or be contained in, one or more of the various computers/clients shown.

Claim 46

Claim 46's "The computer-readable medium according to claim 39, wherein the output data includes an indication of a potential exposure to failure of the computer-based system." is anticipated by Rice, et al., column 55, lines 15-30, where it recites:

> To perform self-modification and error handling requires state information ("introspection"). As a design philosophy, one may attempt to make all state information generated by the Virtual Machine in evaluating agents available to the agents themselves. Metaphorically, to correct errors or make adjustments, one needs to know "Where am I? What went wrong? What was the configuration at the time of the error?" Embodiments of the present invention envision that the virtual machine environment consists of an array of state records and some global variables (userCount, steps, errors, lasterror, etc.). All this information is available in read-only mode

> to the agents. Agents can also query information about the processes that spawned them, enabling context sensitive logic (if this process called me then do x, otherwise do y).

Claim 47

Claim 47's "The computer-readable medium according to claim 39, wherein the output data includes an exposure level to failure of the computer-based system." is anticipated by Rice, et al., column 42, lines 36-52, where it recites:

> Client/Server **codons** allow for many possible **network** architectures, and embodiments are shown with regard to FIG. 20. Referring to FIG. 20, in the example depicted by that Figure, a request to perform a given task is initiated by a client computer X (2002) and sent to Computer A for processing. Using the features envisioned by the present invention, the request could then be routed from computer A to, depending upon load, resources, etc., one or more of computers B, C, D, etc. Assuming (in this example) that Computer D (2010) processes (or helps process) the request, **it is shown that Computer D then passes the result back to client computer X (2002).** In addition to this specific example, FIG. 20 also indicates that more than one client can be involved in a given transaction. It should be understood that the virtual machine environment is contemplated to contain or be contained in, one or more of the various computers/clients shown.

Claim 48

Claim 48's "The computer-readable medium according to claim 47, wherein the output data includes a confidence level of the exposure level." is anticipated by Rice, et al., column 29, lines 35-55, where it recites:

> Still referring to program 902, Step 2 is the cosine operation which, requiring one argument as input, receives the 11 from stack 904, and yields 0.9816, as shown at Step 3 of stack 904. Then, upon encountering Step 3 of the program 902, which is the multiplication operation, the present invention looks for two arguments. Finding only one (i.e., the 0.9816), the present invention's grammar repair generates a second

> argument (which in this example is the number 3) and places it on the stack at step 3 as shown. The multiplication can then commence, yielding the number 2.9448. Thus, the program 902 can then be tested for **fitness** using the numbers generated, as shown on stack 904. It may very well be that the results are unlikely to score well with the fitness function, but at least there will be some result to work with. Embodiments of the current invention contemplate that the specific type of grammar repair function used can differ depending upon the specific codon(s) encountered. Furthermore, it is envisioned that the grammar repair function can vary with each codon and can be set at any time.

Claim 49

Claim 49's "The computer-readable medium according to claim 39, wherein the output data identifies whether the rule engine completed execution of the rule." " is anticipated by Rice, et al., column 42, lines 36-52, where it recites:

> Client/Server **codons** allow for many possible **network** architectures, and embodiments are shown with regard to FIG. 20. Referring to FIG. 20, in the example depicted by that Figure, a request to perform a given task is initiated by a client computer X (2002) and sent to Computer A for processing. Using the features envisioned by the present invention, the request could then be routed from computer A to, depending upon load, resources, etc., one or more of computers B, C, D, etc. Assuming (in this example) that Computer D (2010) processes (or helps process) the request, **it is shown that Computer D then passes the result back to client computer X (2002).** In addition to this specific example, FIG. 20 also indicates that more than one client can be involved in a given transaction. It should be understood that the virtual machine environment is contemplated to contain or be contained in, one or more of the various computers/clients shown.

Claim 50

Claim 50's "The computer-readable medium according to claim 39, wherein the output data identifies that an error occurred during execution of the rule." is anticipated by Rice, et al., column 13, lines 52-67 and column 14, lines 1-11, where it recites:

> "T.sub.-- DATA.sub.-- INDEX" is an implementation-defined integral data type such as "long" whose value is set to the number of items in the return buffer or to a negative number to **indicate that an error occurred**. For example, a "-1" could indicate that an error occurred (usually due to invalid arguments) and the return buffer would be empty. It is further envisioned that there would exist a table of errors which would contain error numbers and error descriptions which could be extended during runtime. A return value of "0" could indicate that the virtual machine function or procedure was successfully executed, but that no values were returned (i.e. the return buffer is empty). Any integer greater than "0" could indicate that the function was successfully executed and the integer returned would indicate the number of items in the return buffer. The value and type of the items could then be accessed in the return buffer. While embodiments of the present invention envision that the argument data types can be determined by the evaluating functions by passing arguments to and from the virtual machines in the form of variants and providing mechanisms for retrieving and converting fundamental and structure types into variants, other possible schemes are also contemplated, including maintaining a table in the virtual machine of the type of arguments each function takes. In the former case, type checking is carried out by the virtual machine functions, in the latter case, the virtual machine itself determines whether the appropriate arguments are available.

Claim 51

Claim 51's "The computer-readable medium according to claim 39, wherein the output data identifies that a side effect occurred during execution of the rule." is anticipated by Rice, et al., column 13, lines 52-67 and column 14, lines 1-11, where it recites:

> "T.sub.-- DATA.sub.-- INDEX" is an implementation-defined integral data type such as "long" whose value is set to the number of items in the return buffer or to a negative number to **indicate that an error occurred**. For example, a "-1" could indicate that an error occurred (usually due to invalid arguments) and the return buffer would be empty. It is further envisioned that there would exist a table of errors which would contain error numbers and error descriptions which could be extended during runtime. A return value of "0" could indicate that the virtual machine function or procedure was successfully executed, but that no values were returned (i.e. the return buffer is empty). Any integer greater than "0" could indicate that the function was successfully executed and the integer returned would indicate the number of items in

> the return buffer. The value and type of the items could then be accessed in the return buffer. While embodiments of the present invention envision that the argument data types can be determined by the evaluating functions by passing arguments to and from the virtual machines in the form of variants and providing mechanisms for retrieving and converting fundamental and structure types into variants, other possible schemes are also contemplated, including maintaining a table in the virtual machine of the type of arguments each function takes. In the former case, type checking is carried out by the virtual machine functions, in the latter case, the virtual machine itself determines whether the appropriate arguments are available.

Claim 52

Claim 52's "a memory comprising a rule engine deployment program that:" is anticipated by Rice, et al., column 26, lines 45-62, where it recites:

> Although the example manipulates programs in their codon form, as a result of the intraconvertiblity of text and codon forms, it is also possible to perform genetic operations on the programs in their textual representation. In the latter case, the genetic operations are simply text operations (e.g. string concatenations). In either case, during the cutting and splicing phase, the representations are treated as data. After **breeding**, the resultant data is, if necessary, converted from text to codons and again treated as an **executable program**.

Claim 52's "extracts a rule information from a subscribed-to rule datatype," is anticipated by Rice, et al., column 1, lines 34-52, where it recites:

> The present invention relates to a fault tolerant and combinatorial software environment. More specifically, the present invention relates to a system, method and medium for providing and facilitating a virtual machine and instruction format that facilitates systematic or arbitrary alteration and recombination of portions of code (e.g., programs) at run-time. Instructions that comprise a language that can be used with the present invention can be individualized, allowing for differentiation of each instruction based on, e.g., usage, origin, authorship, or **any other user-defined tag**. The overall instruction set can be extended to include any object or procedure that can be represented by software. The invention is envisioned for use in applications such as those relating to automatic program creation (e.g. genetic programming), artificial/directed

> evolution of digital designs (graphic, industrial, mechanical, architectural, engineering, etc.), online adaptive software systems, product and process reliability testing, and simulation of biological, non-linear, and probabilistic processes.

Claim 52's "wherein the rule information includes a rule that defines a logic for determining exposure to failure of a computer-based system based on input data about the computer-based system," is anticipated by Rice, et al., column 55, lines 15-30, where it recites:

> To perform self-modification and error handling requires state information ("introspection"). As a design philosophy, one may attempt to make all state information generated by the Virtual Machine in evaluating agents available to the agents themselves. Metaphorically, to correct errors or make adjustments, one needs to know "Where am I? What went wrong? What was the configuration at the time of the error?" Embodiments of the present invention envision that the virtual machine environment consists of an array of state records and some global variables (userCount, steps, errors, lasterror, etc.). All this information is available in read-only mode to the agents. Agents can also query information about the processes that spawned them, enabling context sensitive logic (if this process called me then do x, otherwise do y).

Claim 52's "an identifier of the input data used by the rule, and an identifier of the output data output based on execution of the rule," is anticipated by Rice, et al., column 9, lines 5-14, where it recites:

> Referring now to FIG. 2, two programs (202 and 204) are shown, each having ten steps. The letters shown in each program represent arguments (e.g., constants) that are used as **input** to, e.g., subsequent operations (such as an addition operation), represented by the numbers in the program. Each number, in addition to merely signifying the existence of an operation, indicates the number of arguments required by the operation. (Of course, the numbers could represent any type of entity requiring arguments.)

Further, is anticipated by Rice, et al., column 42, lines 36-52, where it recites:

> Client/Server **codons** allow for many possible **network** architectures, and embodiments are shown with regard to FIG. 20. Referring to FIG. 20, in the example depicted by that Figure, a request to perform a given task is initiated by a client computer X (2002) and sent to Computer A for processing. Using the features envisioned by the present invention, the request could then be routed from computer A to, depending upon load, resources, etc., one or more of computers B, C, D, etc. Assuming (in this example) that Computer D (2010) processes (or helps process) the request, **it is shown that Computer D then passes the result back to client computer X (2002).** In addition to this specific example, FIG. 20 also indicates that more than one client can be involved in a given transaction. It should be understood that the virtual machine environment is contemplated to contain or be contained in, one or more of the various computers/clients shown.

Claim 52's "instantiates a rule engine for executing the rule, the rule engine subscribing to the identified input data and outputting the identified output data responsive to completing processing of the rule, and" is anticipated by Rice, et al., column 9, lines 5-14, where it recites:

> Referring now to FIG. 2, two programs (202 and 204) are shown, each having ten steps. The letters shown in each program represent arguments (e.g., constants) that are used as input to, e.g., subsequent operations (such as an addition operation), represented by the numbers in the program. Each number, in addition to merely signifying the existence of an operation, indicates the number of arguments required by the operation. (Of course, the numbers could represent any type of entity requiring arguments.)

Further, is anticipated by Rice, et al., column 42, lines 36-52, where it recites:

> Client/Server **codons** allow for many possible **network** architectures, and embodiments are shown with regard to FIG. 20. Referring to FIG. 20, in the example depicted by that Figure, a request to perform a given task is initiated by a client computer X (2002) and sent to Computer A for

> processing. Using the features envisioned by the present invention, the request could then be routed from computer A to, depending upon load, resources, etc., one or more of computers B, C, D, etc. Assuming (in this example) that Computer D (2010) processes (or helps process) the request, **it is shown that Computer D then passes the result back to client computer X (2002).** In addition to this specific example, FIG. 20 also indicates that more than one client can be involved in a given transaction. It should be understood that the virtual machine environment is contemplated to contain or be contained in, one or more of the various computers/clients shown.

Claim 52's "deploys the rule engine within a wrapper that encapsulates the rule engine, the wrapper adapted to encapsulate a plurality of rule engines and publish the output data from the rule engine; and" is anticipated by Rice, et al., column 42, lines 36-52, where it recites:

> Client/Server **codons** allow for many possible **network** architectures, and embodiments are shown with regard to FIG. 20. Referring to FIG. 20, in the example depicted by that Figure, a request to perform a given task is initiated by a client computer X (2002) and sent to Computer A for processing. Using the features envisioned by the present invention, the request could then be routed from computer A to, depending upon load, resources, etc., one or more of computers B, C, D, etc. Assuming (in this example) that Computer D (2010) processes (or helps process) the request, **it is shown that Computer D then passes the result back to client computer X (2002).** In addition to this specific example, FIG. 20 also indicates that more than one client can be involved in a given transaction. It should be understood that the virtual machine environment is contemplated to contain or be contained in, one or more of the various computers/clients shown.

Further, is anticipated by Rice, et al., column 11, lines 64-67 and column 12, lines 1-19, where it recites:

> Third, it is also contemplated that one can encapsulate third party compiled code (in real time) into a codon provided that a header file and library file (object file) is available. This involves inspecting the header file, and recording the attributes of the desired "extensions" in a data

> structure maintained by the virtual machine and loading the function's machine instructions into memory. The function (i.e., a function which is part of the third party code) would be assigned a codon in the typical manner, and given a special type (e.g., "user-defined" codon, as mentioned further below). When that codon is encountered, embodiments of the present invention contemplate that the codon is passed to a special **wrapper function** which checks if the appropriate arguments and types are available. If so, the wrapper function extracts the arguments from the virtual machine's data stack, casts them into the appropriate type/form (data types) and places them on the microprocessor's stack. The virtual machine invokes the third party function by transferring control to the memory address where the function was stored previously. The value returned by the function, if any, is converted into the virtual machine's data representation format and placed on its stack similar to any other intrinsic virtual machine function.

Claim 52's "a processing unit that runs the rule engine deployment program." is anticipated by Rice, et al., column 26, lines 45-62, where it recites:

> Although the example manipulates programs in their codon form, as a result of the intraconvertiblity of text and codon forms, it is also possible to perform genetic operations on the programs in their textual representation. In the latter case, the genetic operations are simply text operations (e.g. string concatenations). In either case, during the cutting and splicing phase, the representations are treated as data. After **breeding**, the resultant data is, if necessary, converted from text to codons and again treated as an **executable program**.

Claim 53

Claim 53's "means for extracting a rule information from a subscribed-to rule datatype," is anticipated by Rice, et al., column 1, lines 34-52, where it recites:

> The present invention relates to a fault tolerant and combinatorial software environment. More specifically, the present invention relates to a system, method and medium for providing and facilitating a virtual machine and instruction format that facilitates systematic or arbitrary alteration and recombination of portions of code (e.g., programs) at run-time. Instructions that comprise a language that can be used with the present invention can be individualized, allowing for differentiation of each instruction based on, e.g., usage, origin, authorship, or **any other**

> **user-defined tag**. The overall instruction set can be extended to include any object or procedure that can be represented by software. The invention is envisioned for use in applications such as those relating to automatic program creation (e.g. genetic programming), artificial/directed evolution of digital designs (graphic, industrial, mechanical, architectural, engineering, etc.), online adaptive software systems, product and process reliability testing, and simulation of biological, non-linear, and probabilistic processes.

Claim 53's "wherein the rule information includes a rule that defines a logic for determining exposure to failure of a computer-based system based on input data about the computer-based system," is anticipated by Rice, et al., column 55, lines 15-30, where it recites:

> To perform self-modification and error handling requires state information ("introspection"). As a design philosophy, one may attempt to make all state information generated by the Virtual Machine in evaluating agents available to the agents themselves. Metaphorically, to correct errors or make adjustments, one needs to know "Where am I? What went wrong? What was the configuration at the time of the error?" Embodiments of the present invention envision that the virtual machine environment consists of an array of state records and some global variables (userCount, steps, errors, lasterror, etc.). All this information is available in read-only mode to the agents. Agents can also query information about the processes that spawned them, enabling context sensitive logic (if this process called me then do x, otherwise do y).

Claim 53's "an identifier of the input data used by the rule, and" is anticipated by Rice, et al., column 9, lines 5-14, where it recites:

> Referring now to FIG. 2, two programs (202 and 204) are shown, each having ten steps. The letters shown in each program represent arguments (e.g., constants) that are used as input to, e.g., subsequent operations (such as an addition operation), represented by the numbers in the program. Each number, in addition to merely signifying the existence of an operation, indicates the number of arguments required by the operation. (Of course, the numbers could represent any type of entity requiring arguments.)

Claim 53's "an identifier of the output data output based on execution of the rule;" is anticipated by Rice, et al., column 42, lines 36-52, where it recites:

> Client/Server **codons** allow for many possible **network** architectures, and embodiments are shown with regard to FIG. 20. Referring to FIG. 20, in the example depicted by that Figure, a request to perform a given task is initiated by a client computer X (2002) and sent to Computer A for processing. Using the features envisioned by the present invention, the request could then be routed from computer A to, depending upon load, resources, etc., one or more of computers B, C, D, etc. Assuming (in this example) that Computer D (2010) processes (or helps process) the request, **it is shown that Computer D then passes the result back to client computer X (2002).** In addition to this specific example, FIG. 20 also indicates that more than one client can be involved in a given transaction. It should be understood that the virtual machine environment is contemplated to contain or be contained in, one or more of the various computers/clients shown.

Claim 53's "means for instantiating a rule engine for executing the rule, the rule engine subscribing to the identified input data and outputting the identified output data responsive to completing processing of the rule; and" is anticipated by Rice, et al., column 9, lines 5-14, where it recites:

> Referring now to FIG. 2, two programs (202 and 204) are shown, each having ten steps. The letters shown in each program represent arguments (e.g., constants) that are used as input to, e.g., subsequent operations (such as an addition operation), represented by the numbers in the program. Each number, in addition to merely signifying the existence of an operation, indicates the number of arguments required by the operation. (Of course, the numbers could represent any type of entity requiring arguments.)

Further, is anticipated by Rice, et al., column 42, lines 36-52, where it recites:

> Client/Server **codons** allow for many possible **network** architectures, and embodiments are shown with regard to FIG. 20. Referring to FIG. 20, in the example depicted by that Figure, a request to perform a given task is initiated by a client computer X (2002) and sent to Computer A for processing. Using the features envisioned by the present invention, the request could then be routed from computer A to, depending upon load, resources, etc., one or more of computers B, C, D, etc. Assuming (in this example) that Computer D (2010) processes (or helps process) the request, **it is shown that Computer D then passes the result back to client computer X (2002).** In addition to this specific example, FIG. 20 also indicates that more than one client can be involved in a given transaction. It should be understood that the virtual machine environment is contemplated to contain or be contained in, one or more of the various computers/clients shown.

Claim 53's "means for deploying the rule engine within a wrapper that encapsulates the rule engine, the wrapper adapted to encapsulate a plurality of rule engines and publish the output data from the rule engine." is anticipated by Rice, et al., column 42, lines 36-52, where it recites:

> Client/Server **codons** allow for many possible **network** architectures, and embodiments are shown with regard to FIG. 20. Referring to FIG. 20, in the example depicted by that Figure, a request to perform a given task is initiated by a client computer X (2002) and sent to Computer A for processing. Using the features envisioned by the present invention, the request could then be routed from computer A to, depending upon load, resources, etc., one or more of computers B, C, D, etc. Assuming (in this example) that Computer D (2010) processes (or helps process) the request, **it is shown that Computer D then passes the result back to client computer X (2002).** In addition to this specific example, FIG. 20 also indicates that more than one client can be involved in a given transaction. It should be understood that the virtual machine environment is contemplated to contain or be contained in, one or more of the various computers/clients shown.

Further, is anticipated by Rice, et al., column 11, lines 64-67 and column 12, lines 1-19, where it recites:

> Third, it is also contemplated that one can encapsulate third party compiled code (in real time) into a codon provided that a header file and library file (object file) is available. This involves inspecting the header file, and recording the attributes of the desired "extensions" in a data structure maintained by the virtual machine and loading the function's machine instructions into memory. The function (i.e., a function which is part of the third party code) would be assigned a codon in the typical manner, and given a special type (e.g., "user-defined" codon, as mentioned further below). When that codon is encountered, embodiments of the present invention contemplate that the codon is passed to a special **wrapper function** which checks if the appropriate arguments and types are available. If so, the wrapper function extracts the arguments from the virtual machine's data stack, casts them into the appropriate type/form (data types) and places them on the microprocessor's stack. The virtual machine invokes the third party function by transferring control to the memory address where the function was stored previously. The value returned by the function, if any, is converted into the virtual machine's data representation format and placed on its stack similar to any other intrinsic virtual machine function.

Claim 54

Claim 54's "A method in a data processing system having a rule engine program encapsulated within a wrapper, the method comprising the steps performed by the rule engine program of:" is anticipated by Rice, et al., column 11, lines 64-67 and column 12, lines 1-19, where it recites:

> Third, it is also contemplated that one can encapsulate third party compiled code (in real time) into a codon provided that a header file and library file (object file) is available. This involves inspecting the header file, and recording the attributes of the desired "extensions" in a data structure maintained by the virtual machine and loading the function's machine instructions into memory. The function (i.e., a function which is part of the third party code) would be assigned a codon in the typical manner, and given a special type (e.g., "user-defined" codon, as mentioned further below). When that codon is encountered, embodiments of the present invention contemplate that the codon is passed to a special **wrapper function** which checks if the appropriate arguments and types are available. If so, the wrapper function extracts the arguments from the virtual machine's data stack, casts them into the appropriate type/form (data types) and places them on the microprocessor's stack. The virtual machine invokes the third party function by transferring control to the memory address where the

> function was stored previously. The value returned by the function, if any, is converted into the virtual machine's data representation format and placed on its stack similar to any other intrinsic virtual machine function.

Claim 54's "receiving subscribed-to input data about a computer-based system;" is anticipated by Rice, et al., column 9, lines 5-14, where it recites:

> Referring now to FIG. 2, two programs (202 and 204) are shown, each having ten steps. The letters shown in each program represent arguments (e.g., constants) that are used as input to, e.g., subsequent operations (such as an addition operation), represented by the numbers in the program. Each number, in addition to merely signifying the existence of an operation, indicates the number of arguments required by the operation. (Of course, the numbers could represent any type of entity requiring arguments.)

Claim 54's "executing a rule that defines a logic for determining exposure to failure of the computer-based system based on the received input data; and" is anticipated by Rice, et al., column 55, lines 15-30, where it recites:

> To perform self-modification and error handling requires state information ("introspection"). As a design philosophy, one may attempt to make all state information generated by the Virtual Machine in evaluating agents available to the agents themselves. Metaphorically, to correct errors or make adjustments, one needs to know "Where am I? What went wrong? What was the configuration at the time of the error?" Embodiments of the present invention envision that the virtual machine environment consists of an array of state records and some global variables (userCount, steps, errors, lasterror, etc.). All this information is available in read-only mode to the agents. Agents can also query information about the processes that spawned them, enabling context sensitive logic (if this process called me then do x, otherwise do y).

Claim 54's "outputting an output data responsive to a determination that there is an exposure to failure." is anticipated by Rice, et al., column 55, lines 15-30, where it recites:

> To perform self-modification and error handling requires state information ("introspection"). As a design philosophy, one may attempt to make all state information generated by the Virtual Machine in evaluating agents available to the agents themselves. Metaphorically, to correct errors or make adjustments, one needs to know "Where am I? What went wrong? What was the configuration at the time of the error?" Embodiments of the present invention envision that the virtual machine environment consists of an array of state records and some global variables (userCount, steps, errors, lasterror, etc.). All this information is available in read-only mode to the agents. Agents can also query information about the processes that spawned them, enabling context sensitive logic (if this process called me then do x, otherwise do y).

Claim 55

Claim 55's "prior to executing the rule, executing a preliminary rule to determine whether the rule is to be executed." is anticipated by Rice, et al., column 11, lines 64-67 and column 12, lines 1-19, where it recites:

> Third, it is also contemplated that one can encapsulate third party compiled code (in real time) into a codon provided that a header file and library file (object file) is available. This involves inspecting the header file, and recording the attributes of the desired "extensions" in a data structure maintained by the virtual machine and loading the function's machine instructions into memory. The function (i.e., a function which is part of the third party code) would be assigned a codon in the typical manner, and given a special type (e.g., "user-defined" codon, as mentioned further below). When that codon is encountered, embodiments of the present invention contemplate that the codon is passed to a special **wrapper function** which **checks if the appropriate arguments and types are available**. If so, the wrapper function extracts the arguments from the virtual machine's data stack, **casts** them into the appropriate type/form (data types) and **places** them on the microprocessor's stack. The virtual machine invokes the third party function by transferring control to the memory address where the

function was stored previously. The value returned by the function, if any, is converted into the virtual machine's data representation format and placed on its stack similar to any other intrinsic virtual machine function.

Claim 56

Claim 56's "The method according to claim 54, wherein the output data from the rule engine is subscribed to by another rule engine." is anticipated by Rice, et al., column 42, lines 36-52, where it recites:

Client/Server **codons** allow for many possible **network** architectures, and embodiments are shown with regard to FIG. 20. Referring to FIG. 20, in the example depicted by that Figure, a request to perform a given task is initiated by a client computer X (2002) and sent to Computer A for processing. Using the features envisioned by the present invention, the request could then be routed from computer A to, depending upon load, resources, etc., one or more of computers B, C, D, etc. Assuming (in this example) that Computer D (2010) processes (or helps process) the request, **it is shown that Computer D then passes the result back to client computer X (2002).** In addition to this specific example, FIG. 20 also indicates that more than one client can be involved in a given transaction. It should be understood that the virtual machine environment is contemplated to contain or be contained in, one or more of the various computers/clients shown.

Claim 57

Claim 57's "The method according to claim 54, wherein the output data includes an indication of a potential exposure to failure of the computer-based system." is anticipated by Rice, et al., column 55, lines 15-30, where it recites:

To perform self-modification and error handling requires state information ("introspection"). As a design philosophy, one may attempt to make all state information generated by the Virtual Machine in evaluating agents available to the agents themselves. Metaphorically, to correct errors or make adjustments, one needs to know "Where am I? What went wrong? What was the configuration at the time of the error?" Embodiments of the

present invention envision that the virtual machine environment consists of an array of state records and some global variables (userCount, steps, errors, lasterror, etc.). All this information is available in read-only mode to the agents. Agents can also query information about the processes that spawned them, enabling context sensitive logic (if this process called me then do x, otherwise do y).

Claim 58

Claim 58's "The method according to claim 54, wherein the output data includes an exposure level to failure of the computer-based system." is anticipated by Rice, et al., column 42, lines 36-52, where it recites:

Client/Server **codons** allow for many possible **network** architectures, and embodiments are shown with regard to FIG. 20. Referring to FIG. 20, in the example depicted by that Figure, a request to perform a given task is initiated by a client computer X (2002) and sent to Computer A for processing. Using the features envisioned by the present invention, the request could then be routed from computer A to, depending upon load, resources, etc., one or more of computers B, C, D, etc. Assuming (in this example) that Computer D (2010) processes (or helps process) the request, **it is shown that Computer D then passes the result back to client computer X (2002).** In addition to this specific example, FIG. 20 also indicates that more than one client can be involved in a given transaction. It should be understood that the virtual machine environment is contemplated to contain or be contained in, one or more of the various computers/clients shown.

Claim 59

Claim 59's "The method according to claim 58, wherein the output data includes a confidence level of the exposure level." is anticipated by Rice, et al., column 29, lines 35-55, where it recites:

Still referring to program 902, Step 2 is the cosine operation which, requiring one argument as input, receives the 11 from stack 904, and yields 0.9816, as shown at Step 3 of stack 904. Then, upon encountering

> Step 3 of the program 902, which is the multiplication operation, the present invention looks for two arguments. Finding only one (i.e., the 0.9816), the present invention's grammar repair generates a second argument (which in this example is the number 3) and places it on the stack at step 3 as shown. The multiplication can then commence, yielding the number 2.9448. Thus, the program 902 can then be tested for **fitness** using the numbers generated, as shown on stack 904. It may very well be that the results are unlikely to score well with the fitness function, but at least there will be some result to work with. Embodiments of the current invention contemplate that the specific type of grammar repair function used can differ depending upon the specific codon(s) encountered. Furthermore, it is envisioned that the grammar repair function can vary with each codon and can be set at any time.

Claim 60

Claim 60's "The method according to claim 54, wherein the output data identifies whether the rule engine completed execution of the rule." " is anticipated by Rice, et al., column 42, lines 36-52, where it recites:

> Client/Server **codons** allow for many possible **network** architectures, and embodiments are shown with regard to FIG. 20. Referring to FIG. 20, in the example depicted by that Figure, a request to perform a given task is initiated by a client computer X (2002) and sent to Computer A for processing. Using the features envisioned by the present invention, the request could then be routed from computer A to, depending upon load, resources, etc., one or more of computers B, C, D, etc. Assuming (in this example) that Computer D (2010) processes (or helps process) the request, **it is shown that Computer D then passes the result back to client computer X (2002).** In addition to this specific example, FIG. 20 also indicates that more than one client can be involved in a given transaction. It should be understood that the virtual machine environment is contemplated to contain or be contained in, one or more of the various computers/clients shown.

Claim 61

Claim 61's "The method according to claim 54, wherein the output data identifies that an error occurred during execution of the rule." is anticipated by Rice, et al., column 13, lines 52-67 and column 14, lines 1-11, where it recites:

> "T.sub.-- DATA.sub.-- INDEX" is an implementation-defined integral data type such as "long" whose value is set to the number of items in the return buffer or to a negative number to **indicate that an error occurred**. For example, a "-1" could indicate that an error occurred (usually due to invalid arguments) and the return buffer would be empty. It is further envisioned that there would exist a table of errors which would contain error numbers and error descriptions which could be extended during runtime. A return value of "0" could indicate that the virtual machine function or procedure was successfully executed, but that no values were returned (i.e. the return buffer is empty). Any integer greater than "0" could indicate that the function was successfully executed and the integer returned would indicate the number of items in the return buffer. The value and type of the items could then be accessed in the return buffer. While embodiments of the present invention envision that the argument data types can be determined by the evaluating functions by passing arguments to and from the virtual machines in the form of variants and providing mechanisms for retrieving and converting fundamental and structure types into variants, other possible schemes are also contemplated, including maintaining a table in the virtual machine of the type of arguments each function takes. In the former case, type checking is carried out by the virtual machine functions, in the latter case, the virtual machine itself determines whether the appropriate arguments are available.

Claim 62

Claim 62's "The method according to claim 54, wherein the output data identifies that a side effect occurred during execution of the rule." is anticipated by Rice, et al., column 13, lines 52-67 and column 14, lines 1-11, where it recites:

> "T.sub.-- DATA.sub.-- INDEX" is an implementation-defined integral data type such as "long" whose value is set to the number of items in the return buffer or to a negative number to **indicate that an error occurred**. For example, a "-1" could indicate that an error occurred (usually due to invalid arguments) and the return buffer would be empty. It is further envisioned that there would exist a table of errors which would contain error numbers and error descriptions which could be extended during runtime. A return value of "0" could indicate that the virtual machine function or procedure was successfully executed, but that no values were returned (i.e. the return buffer is empty). Any integer greater than "0" could indicate that the function was successfully executed and the integer returned would indicate the number of items in

> the return buffer. The value and type of the items could then be accessed in the return buffer. While embodiments of the present invention envision that the argument data types can be determined by the evaluating functions by passing arguments to and from the virtual machines in the form of variants and providing mechanisms for retrieving and converting fundamental and structure types into variants, other possible schemes are also contemplated, including maintaining a table in the virtual machine of the type of arguments each function takes. In the former case, type checking is carried out by the virtual machine functions, in the latter case, the virtual machine itself determines whether the appropriate arguments are available.

Claim 63

Claim 63's "receiving subscribed-to input data about a computer-based system;" is anticipated by Rice, et al., column 9, lines 5-14, where it recites:

> Referring now to FIG. 2, two programs (202 and 204) are shown, each having ten steps. The letters shown in each program represent arguments (e.g., constants) that are used as input to, e.g., subsequent operations (such as an addition operation), represented by the numbers in the program. Each number, in addition to merely signifying the existence of an operation, indicates the number of arguments required by the operation. (Of course, the numbers could represent any type of entity requiring arguments.)

Claim 63's "executing a rule that defines a logic for determining exposure to failure of the computer-based system based on the received input data; and" is anticipated by Rice, et al., column 55, lines 15-30, where it recites:

> To perform self-modification and error handling requires state information ("introspection"). As a design philosophy, one may attempt to make all state information generated by the Virtual Machine in evaluating agents available to the agents themselves. Metaphorically, to correct errors or make adjustments, one needs to know "Where am I? What went wrong? What was the configuration at the time of the error?" Embodiments of the present invention envision that the virtual machine environment consists of an array of state records and some global variables (userCount, steps, errors, lasterror, etc.). All this information is available in read-only mode

> to the agents. Agents can also query information about the processes that spawned them, enabling context sensitive logic (if this process called me then do x, otherwise do y).

Claim 63's "outputting an output data responsive to a determination that there is an exposure to failure." is anticipated by Rice, et al., column 55, lines 15-30, where it recites:

> To perform self-modification and error handling requires state information ("introspection"). As a design philosophy, one may attempt to make all state information generated by the Virtual Machine in evaluating agents available to the agents themselves. Metaphorically, to correct errors or make adjustments, one needs to know "Where am I? What went wrong? What was the configuration at the time of the error?" Embodiments of the present invention envision that the virtual machine environment consists of an array of state records and some global variables (userCount, steps, errors, lasterror, etc.). All this information is available in read-only mode to the agents. Agents can also query information about the processes that spawned them, enabling context sensitive logic (if this process called me then do x, otherwise do y).

Claim 64

Claim 64's "prior to executing the rule, executing a preliminary rule to determine whether the rule is to be executed." is anticipated by Rice, et al., column 11, lines 64-67 and column 12, lines 1-19, where it recites:

> Third, it is also contemplated that one can encapsulate third party compiled code (in real time) into a codon provided that a header file and library file (object file) is available. This involves inspecting the header file, and recording the attributes of the desired "extensions" in a data structure maintained by the virtual machine and loading the function's machine instructions into memory. The function (i.e., a function which is part of the third party code) would be assigned a codon in the typical manner, and given a special type (e.g., "user-defined" codon, as mentioned further below). When that codon is encountered, embodiments of the present invention contemplate that the codon is passed to a special **wrapper function** which **checks if the appropriate**

> **arguments and types are available**. If so, the wrapper function extracts the arguments from the virtual machine's data stack, **casts** them into the appropriate type/form (data types) and **places** them on the microprocessor's stack. The virtual machine invokes the third party function by transferring control to the memory address where the function was stored previously. The value returned by the function, if any, is converted into the virtual machine's data representation format and placed on its stack similar to any other intrinsic virtual machine function.

Claim 65

Claim 65's "The computer-readable medium according to claim 63, wherein the output data from the rule engine is subscribed to by another rule engine." is anticipated by Rice, et al., column 42, lines 36-52, where it recites:

> Client/Server **codons** allow for many possible **network** architectures, and embodiments are shown with regard to FIG. 20. Referring to FIG. 20, in the example depicted by that Figure, a request to perform a given task is initiated by a client computer X (2002) and sent to Computer A for processing. Using the features envisioned by the present invention, the request could then be routed from computer A to, depending upon load, resources, etc., one or more of computers B, C, D, etc. Assuming (in this example) that Computer D (2010) processes (or helps process) the request, **it is shown that Computer D then passes the result back to client computer X (2002).** In addition to this specific example, FIG. 20 also indicates that more than one client can be involved in a given transaction. It should be understood that the virtual machine environment is contemplated to contain or be contained in, one or more of the various computers/clients shown.

Claim 66

Claim 66's "The computer-readable medium according to claim 63, wherein the output data includes an indication of a potential exposure to failure of the computer-based system." is anticipated by Rice, et al., column 55, lines 15-30, where it recites:

> To perform self-modification and error handling requires state information ("introspection"). As a design philosophy, one may attempt to make all state information generated by the Virtual Machine in evaluating agents available to the agents themselves. Metaphorically, to correct errors or make adjustments, one needs to know "Where am I? What went wrong? What was the configuration at the time of the error?" Embodiments of the present invention envision that the virtual machine environment consists of an array of state records and some global variables (userCount, steps, errors, lasterror, etc.). All this information is available in read-only mode to the agents. Agents can also query information about the processes that spawned them, enabling context sensitive logic (if this process called me then do x, otherwise do y).

Claim 67

Claim 67's "The computer-readable medium according to claim 63, wherein the output data includes an exposure level to failure of the computer-based system." is anticipated by Rice, et al., column 42, lines 36-52, where it recites:

> Client/Server **codons** allow for many possible **network** architectures, and embodiments are shown with regard to FIG. 20. Referring to FIG. 20, in the example depicted by that Figure, a request to perform a given task is initiated by a client computer X (2002) and sent to Computer A for processing. Using the features envisioned by the present invention, the request could then be routed from computer A to, depending upon load, resources, etc., one or more of computers B, C, D, etc. Assuming (in this example) that Computer D (2010) processes (or helps process) the request, **it is shown that Computer D then passes the result back to client computer X (2002).** In addition to this specific example, FIG. 20 also indicates that more than one client can be involved in a given transaction. It should be understood that the virtual machine environment is contemplated to contain or be contained in, one or more of the various computers/clients shown.

Claim 68

Claim 68's "The computer-readable medium according to claim 67, wherein the output data includes a confidence level of the exposure level." is anticipated by Rice, et al., column 29, lines 35-55, where it recites:

> Still referring to program 902, Step 2 is the cosine operation which, requiring one argument as input, receives the 11 from stack 904, and yields 0.9816, as shown at Step 3 of stack 904. Then, upon encountering Step 3 of the program 902, which is the multiplication operation, the present invention looks for two arguments. Finding only one (i.e., the 0.9816), the present invention's grammar repair generates a second argument (which in this example is the number 3) and places it on the stack at step 3 as shown. The multiplication can then commence, yielding the number 2.9448. Thus, the program 902 can then be tested for **fitness** using the numbers generated, as shown on stack 904. It may very well be that the results are unlikely to score well with the fitness function, but at least there will be some result to work with. Embodiments of the current invention contemplate that the specific type of grammar repair function used can differ depending upon the specific codon(s) encountered. Furthermore, it is envisioned that the grammar repair function can vary with each codon and can be set at any time.

Claim 69

Claim 69's "The computer-readable medium according to claim 63, wherein the output data identifies whether the rule engine completed execution of the rule." " is anticipated by Rice, et al., column 42, lines 36-52, where it recites:

> Client/Server **codons** allow for many possible **network** architectures, and embodiments are shown with regard to FIG. 20. Referring to FIG. 20, in the example depicted by that Figure, a request to perform a given task is initiated by a client computer X (2002) and sent to Computer A for processing. Using the features envisioned by the present invention, the request could then be routed from computer A to, depending upon load, resources, etc., one or more of computers B, C, D, etc. Assuming (in this example) that Computer D (2010) processes (or helps process) the request, **it is shown that Computer D then passes the result back to client computer X (2002).** In addition to this specific example, FIG. 20 also indicates that more than one client can be involved in a given transaction. It should be understood that the virtual machine environment is contemplated to contain or be contained in, one or more of the various computers/clients shown.

Claim 70

Claim 70's "The computer-readable medium according to claim 63, wherein the output data identifies that an error occurred during execution of the rule." is anticipated by Rice, et al., column 13, lines 52-67 and column 14, lines 1-11, where it recites:

> "T.sub.-- DATA.sub.-- INDEX" is an implementation-defined integral data type such as "long" whose value is set to the number of items in the return buffer or to a negative number to **indicate that an error occurred**. For example, a "-1" could indicate that an error occurred (usually due to invalid arguments) and the return buffer would be empty. It is further envisioned that there would exist a table of errors which would contain error numbers and error descriptions which could be extended during runtime. A return value of "0" could indicate that the virtual machine function or procedure was successfully executed, but that no values were returned (i.e. the return buffer is empty). Any integer greater than "0" could indicate that the function was successfully executed and the integer returned would indicate the number of items in the return buffer. The value and type of the items could then be accessed in the return buffer. While embodiments of the present invention envision that the argument data types can be determined by the evaluating functions by passing arguments to and from the virtual machines in the form of variants and providing mechanisms for retrieving and converting fundamental and structure types into variants, other possible schemes are also contemplated, including maintaining a table in the virtual machine of the type of arguments each function takes. In the former case, type checking is carried out by the virtual machine functions, in the latter case, the virtual machine itself determines whether the appropriate arguments are available.

Claim 71

Claim 71's "The computer-readable medium according to claim 63, wherein the output data identifies that a side effect occurred during execution of the rule." is anticipated by Rice, et al., column 13, lines 52-67 and column 14, lines 1-11, where it recites:

> "T.sub.-- DATA.sub.-- INDEX" is an implementation-defined integral data type such as "long" whose value is set to the number of items in the return buffer or to a negative number to **indicate that an error**

occurred. For example, a "-1" could indicate that an error occurred (usually due to invalid arguments) and the return buffer would be empty. It is further envisioned that there would exist a table of errors which would contain error numbers and error descriptions which could be extended during runtime. A return value of "0" could indicate that the virtual machine function or procedure was successfully executed, but that no values were returned (i.e. the return buffer is empty). Any integer greater than "0" could indicate that the function was successfully executed and the integer returned would indicate the number of items in the return buffer. The value and type of the items could then be accessed in the return buffer. While embodiments of the present invention envision that the argument data types can be determined by the evaluating functions by passing arguments to and from the virtual machines in the form of variants and providing mechanisms for retrieving and converting fundamental and structure types into variants, other possible schemes are also contemplated, including maintaining a table in the virtual machine of the type of arguments each function takes. In the former case, type checking is carried out by the virtual machine functions, in the latter case, the virtual machine itself determines whether the appropriate arguments are available.

Claim 72

Claim 72's "a memory comprising a rule engine program encapsulated within a wrapper that receives subscribed-to input data about a computer-based system," is anticipated by Rice, et al., column 11, lines 64-67 and column 12, lines 1-19, where it recites:

Third, it is also contemplated that one can encapsulate third party compiled code (in real time) into a codon provided that a header file and library file (object file) is available. This involves inspecting the header file, and recording the attributes of the desired "extensions" in a data structure maintained by the virtual machine and loading the function's machine instructions into memory. The function (i.e., a function which is part of the third party code) would be assigned a codon in the typical manner, and given a special type (e.g., "user-defined" codon, as mentioned further below). When that codon is encountered, embodiments of the present invention contemplate that the codon is passed to a special **wrapper function** which checks if the appropriate arguments and types are available. If so, the wrapper function extracts the arguments from the virtual machine's data stack, casts them into the appropriate type/form (data types) and places them on the microprocessor's stack. The virtual machine invokes the third party

> function by transferring control to the memory address where the function was stored previously. The value returned by the function, if any, is converted into the virtual machine's data representation format and placed on its stack similar to any other intrinsic virtual machine function.

Claim 72's "executes a rule that defines a logic for determining exposure to failure of the computer-based system based on the received input data, and" is anticipated by Rice, et al., column 55, lines 15-30, where it recites:

> To perform self-modification and error handling requires state information ("introspection"). As a design philosophy, one may attempt to make all state information generated by the Virtual Machine in evaluating agents available to the agents themselves. Metaphorically, to correct errors or make adjustments, one needs to know "Where am I? What went wrong? What was the configuration at the time of the error?" Embodiments of the present invention envision that the virtual machine environment consists of an array of state records and some global variables (userCount, steps, errors, lasterror, etc.). All this information is available in read-only mode to the agents. Agents can also query information about the processes that spawned them, enabling context sensitive logic (if this process called me then do x, otherwise do y).

Claim 72's "outputs an output data responsive to a determination that there is an exposure to failure; and" is anticipated by Rice, et al., column 55, lines 15-30, where it recites:

> To perform self-modification and error handling requires state information ("introspection"). As a design philosophy, one may attempt to make all state information generated by the Virtual Machine in evaluating agents available to the agents themselves. Metaphorically, to correct errors or make adjustments, one needs to know "Where am I? What went wrong? What was the configuration at the time of the error?" Embodiments of the present invention envision that the virtual machine environment consists of an array of state records and some global variables (userCount, steps, errors, lasterror, etc.). All this information is available in read-only mode to the agents. Agents can also query information about the processes that spawned them, enabling context sensitive logic (if this process called me then do x, otherwise do y).

Claim 72's "a processing unit that runs the rule engine program." is anticipated by Rice, et al., column 91, lines 27-41, where it recites:

> In general, it should be emphasized that the various components of embodiments of the present invention can be implemented in hardware, software or a combination thereof. In such embodiments, the various components and steps would be implemented in hardware and/or software to perform the functions of the present invention. Any presently available or future developed computer software language and/or hardware components can be employed in such embodiments of the present invention. For example, at least some of the functionality mentioned above could be implemented using the C, C++, Visual Basic or any assembly language appropriate in view of the processor(s) being used. It could also be written in an interpretive environment such as Java and transported to multiple destinations to various users.

Claim 73

Claim 73's "A data processing system having a rule engine encapsulated within a wrapper, the data processing system comprising:" is anticipated by Rice, et al., column 11, lines 64-67 and column 12, lines 1-19, where it recites:

> Third, it is also contemplated that one can encapsulate third party compiled code (in real time) into a codon provided that a header file and library file (object file) is available. This involves inspecting the header file, and recording the attributes of the desired "extensions" in a data structure maintained by the virtual machine and loading the function's machine instructions into memory. The function (i.e., a function which is part of the third party code) would be assigned a codon in the typical manner, and given a special type (e.g., "user-defined" codon, as mentioned further below). When that codon is encountered, embodiments of the present invention contemplate that the codon is passed to a special **wrapper function** which checks if the appropriate arguments and types are available. If so, the wrapper function extracts the arguments from the virtual machine's data stack, casts them into the appropriate type/form (data types) and places them on the microprocessor's stack. The virtual machine invokes the third party function by transferring control to the memory address where the

> function was stored previously. The value returned by the function, if any, is converted into the virtual machine's data representation format and placed on its stack similar to any other intrinsic virtual machine function.

Claim 73's "means for receiving subscribed-to input data about a computer-based system;" is anticipated by Rice, et al., column 9, lines 5-14, where it recites:

> Referring now to FIG. 2, two programs (202 and 204) are shown, each having ten steps. The letters shown in each program represent arguments (e.g., constants) that are used as input to, e.g., subsequent operations (such as an addition operation), represented by the numbers in the program. Each number, in addition to merely signifying the existence of an operation, indicates the number of arguments required by the operation. (Of course, the numbers could represent any type of entity requiring arguments.)

Claim 73's "means for executing a rule that defines a logic for determining exposure to failure of the computer-based system based on the received input data; and" is anticipated by Rice, et al., column 55, lines 15-30, where it recites:

> To perform self-modification and error handling requires state information ("introspection"). As a design philosophy, one may attempt to make all state information generated by the Virtual Machine in evaluating agents available to the agents themselves. Metaphorically, to correct errors or make adjustments, one needs to know "Where am I? What went wrong? What was the configuration at the time of the error?" Embodiments of the present invention envision that the virtual machine environment consists of an array of state records and some global variables (userCount, steps, errors, lasterror, etc.). All this information is available in read-only mode to the agents. Agents can also query information about the processes that spawned them, enabling context sensitive logic (if this process called me then do x, otherwise do y).

Claim 73's "means for outputting an output data responsive to a determination that there is an exposure to failure." is anticipated by Rice, et al., column 55, lines 15-30, where it recites:

> To perform self-modification and error handling requires state information ("introspection"). As a design philosophy, one may attempt to make all state information generated by the Virtual Machine in evaluating agents available to the agents themselves. Metaphorically, to correct errors or make adjustments, one needs to know "Where am I? What went wrong? What was the configuration at the time of the error?" Embodiments of the present invention envision that the virtual machine environment consists of an array of state records and some global variables (userCount, steps, errors, lasterror, etc.). All this information is available in read-only mode to the agents. Agents can also query information about the processes that spawned them, enabling context sensitive logic (if this process called me then do x, otherwise do y).

Claim 74

Claim 74's "an exposure level to failure of a computer-based system and an identifier of the computer-based system," is anticipated by Rice, et al., column 42, lines 36-52, where it recites:

> Client/Server **codons** allow for many possible **network** architectures, and embodiments are shown with regard to FIG. 20. Referring to FIG. 20, in the example depicted by that Figure, a request to perform a given task is initiated by a client computer X (2002) and sent to Computer A for processing. Using the features envisioned by the present invention, the request could then be routed from computer A to, depending upon load, resources, etc., one or more of computers B, C, D, etc. Assuming (in this example) that Computer D (2010) processes (or helps process) the request, **it is shown that Computer D then passes the result back to client computer X (2002).** In addition to this specific example, FIG. 20 also indicates that more than one client can be involved in a given transaction. It should be understood that the virtual machine environment is contemplated to contain or be contained in, one or more of the various computers/clients shown.

Claim 74's "the program receiving a subscribed-to input data about the computer-based system," is anticipated by Rice, et al., column 9, lines 5-14, where it recites:

> Referring now to FIG. 2, two programs (202 and 204) are shown, each having ten steps. The letters shown in each program represent arguments (e.g., constants) that are used as input to, e.g., subsequent operations (such as an addition operation), represented by the numbers in the program. Each number, in addition to merely signifying the existence of an operation, indicates the number of arguments required by the operation. (Of course, the numbers could represent any type of entity requiring arguments.)

Claim 74's "executing a rule that defines a logic for determining exposure to failure of the computer-based system based on the received input data; and" is anticipated by Rice, et al., column 55, lines 15-30, where it recites:

> To perform self-modification and error handling requires state information ("introspection"). As a design philosophy, one may attempt to make all state information generated by the Virtual Machine in evaluating agents available to the agents themselves. Metaphorically, to correct errors or make adjustments, one needs to know "Where am I? What went wrong? What was the configuration at the time of the error?" Embodiments of the present invention envision that the virtual machine environment consists of an array of state records and some global variables (userCount, steps, errors, lasterror, etc.). All this information is available in read-only mode to the agents. Agents can also query information about the processes that spawned them, enabling context sensitive logic (if this process called me then do x, otherwise do y).

Claim 74's "calculating the exposure level responsive to a determination that there is an exposure to failure." is anticipated by Rice, et al., column 55, lines 15-30, where it recites:

> To perform self-modification and error handling requires state information ("introspection"). As a design philosophy, one may attempt to make all state information generated by the Virtual Machine in evaluating agents available to the agents themselves. Metaphorically, to correct errors or make adjustments, one needs to know "Where am I? What went wrong? What was the configuration at the time of the error?" Embodiments of the present invention envision that the virtual machine environment consists of an array of state records and some global variables (userCount, steps, errors, lasterror, etc.). All this information is available in read-only mode to the agents. Agents can also query information about the processes that spawned them, enabling context sensitive logic (if this process called me then do x, otherwise do y).

Conclusion

The prior art made of record and not relied upon is considered pertinent to Applicant's disclosure. Specifically:

A. Gleichauf et al. (U.S. Patent Number 6,324,656 B1; dated 27 NOV 2001; class 714; subclass 037) discloses a system and method for rules-driven multi-phase network vulnerability assessment.

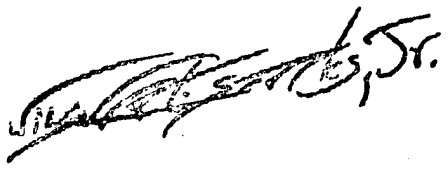
B. Gleichauf et al. (U.S. Patent Number 6,301,668 B1; dated 09 OCT 2001; class 726; subclass 025) discloses a method and system for adaptive network security using network vulnerability assessment.

C. Francone, et al. (U.S. Patent Number 5,946,673 A; dated 31 AUG 1999; class 706; subclass 013) discloses a computer implemented machine learning and control system.

Any inquiry concerning this communication or earlier communications from the Examiner should be directed to Wilbert L. Starks, Jr. whose telephone number is (571) 272-3691.

Alternatively, inquiries may be directed to the following:

S. P. E. David Vincent	**(571) 272-3080**
Official (FAX)	**(571) 273-8300**



Wilbert L. Starks, Jr.
Primary Examiner
Art Unit 2129

WLS

18 JAN 2008